Ervaring met het gedistribueerde bedrijfssysteem Amoeba

Andrew S. Tanenbaum, Robbert van Renesse, Hans van Staveren, Gregory J. Sharp, Sape J. Mullender, Jack Jansen en Guido van Rossum

Het gedistribueerde bedrijfssysteem Amoeba werd in de afgelopen acht jaar ontwikkeld en gebruikt. In dit artikel beschrijven we het huidige systeem en onze ervaringen ermee – wat we goed deden, maar ook wat we fout deden. De dingen die we goed deden, waren onder andere het baseren van het systeem op objecten, het gebruiken van één enkel uniform mechanisme (te weten capabilities) om ze te benoemen en beschermen op een manier die onafhankelijk is van hun lokatie, en het ontwerpen van een volledig nieuw en zeer snel filesysteem.

De dingen die we fout deden, waren onder andere het niet scheppen van de mogelijkheid threads voortijdig te onderbreken, het feit dat we aanvankelijk een eigen window-systeem bouwden en het in eerste instantie weglaten van een faciliteit voor multicasting.

1 Inleiding

Het Amoeba-project is een onderzoek, gericht op het verkrijgen van begrip over de manier waarop men verschillende computers naadloos aan elkaar kan verbinden.[15,16,26,28,32] De basisgedachte is gebruikers de illusie te geven van één krachtig timesharing-systeem, terwijl het systeem in feite is geïmplementeerd op een verzameling machines, mogelijk gedistribueerd over verschillende landen. Dit onderzoek heeft geleid tot het ontwerp en de implementatie van Amoeba, een gedistribueerd bedrijfssysteem, dat wordt gebruikt als een prototype en als onderzoeksmiddel. In dit artikel zullen we de huidige toestand van het systeem (Amoeba 4.0) beschrijven en aandacht schenken aan een aantal lessen die we hebben geleerd tijdens het ontwerp en gebruik in de afgelopen acht jaar. We zullen tevens bespreken hoe deze ervaring onze plannen voor de volgende versie, Amoeba 5.0, heeft beïnvloed.

Amoeba werd oorspronkelijk ontworpen en geïmplementeerd aan de Vrije Universiteit in Amsterdam en wordt op het ogenblik ontwikkeld in een gezamenlijk verband met het Centrum voor Wiskunde en Informatica (CWI), ook in Amsterdam. Het hoofddoel van dit werk is het bouwen van een gedistribueerd systeem dat *transparant* is voor de gebruikers. Dit concept kan het best worden geïllustreerd door het te plaatsen tegenover een bedrijfssysteem in een netwerk, waarbij elke machine zijn eigen identiteit behoudt. In een netwerk-bedrijfssysteem logt elke gebruiker in op een specifieke machine, zijn 'home machine'. Als een programma wordt gestart, draait het op de home machine, tenzij de gebruiker een expliciete opdracht geeft om het ergens anders te draaien. Op een vergelijkbare manier zijn files lokaal, tenzij een elders gesitueerd ('remote') filesysteem expliciet wordt aangekoppeld of files expliciet worden gekopieerd. In het kort komt het erop neer dat de gebruiker zich duidelijk bewust is van het feit dat er verscheidene onafhankelijke computers bestaan en van het feit dat hij daar expliciet mee moet omgaan.

In contrast hiermee staat een transparant gedistribueerd systeem. In zo'n systeem loggen gebruikers in feite in op het systeem als geheel en niet op een specifieke machine. Als een programma wordt gedraaid, beslist het systeem, en niet de gebruiker, wat de beste plaats is om het te draaien. De gebruiker is zich niet eens bewust van de keuze. Tot slot is er één enkel filesysteem voor het hele systeem. De files in een directory kunnen zich bevinden op verschillende machines, mogelijk zelfs in verschillende landen. Er bestaan geen concepten als verplaatsen, uploaden of downloaden van files of het aankoppelen van remote filesystemen. De positie van een file in de hiërarchie van directories heeft geen relatie met zijn lokatie.

De rest van dit artikel zal Amoeba beschrijven en de lessen die we leerden door het systeem te bouwen. In de volgende paragraaf geven we een technisch overzicht van de huidige toestand van Amoeba. Omdat Amoeba het client-server-model gebruikt, zullen we daarna een aantal van de belangrijkste servers bespreken die we tot dusverre hebben geïmplementeerd. Dit wordt gevolgd door een beschrijving van de wijze waarop wordt omgegaan met wide-area netwerken. Dan zullen we een aantal applicaties die op Amoeba draaien, beschrijven. Uit metingen is gebleken dat Amoeba snel is, dus zullen we gegevens presenteren die dit feit illustreren. Daarna zullen we de successen die we behaalden en de mislukkingen waarmee we werden geconfronteerd, bespreken, zodat anderen kunnen profiteren van de ideeën die goed uitpakten, en de ideeën die slecht uitpakten kunnen vermijden. We besluiten met een zeer korte vergelijking van Amoeba met andere systemen.

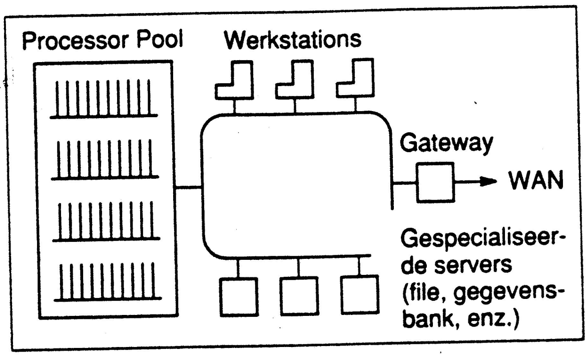
2 Technisch overzicht van Amoeba

Voordat de software beschreven wordt, is het de moeite waard iets te zeggen over de architectuur van het systeem waarop Amoeba draait.

2.1 Architectuur van het systeem

De architectuur van Amoeba bestaat uit vier hoofdcomponenten, zoals te zien is in figuur 1. Ten eerste zijn er werkstations, één per gebruiker, waar gebruikers taken kunnen uitvoeren die snelle interactieve respons vereisen, zoals tekst-editing. De werkstations hebben geen van alle disk-units. Ze worden hoofdzakelijk gebruikt als intelligente terminals die window-management doen en niet als computers voor het draaien van complexe programma's van de gebruiker. Op het ogenblik gebruiken we Sun-3s en VAX-stations als werkstations. In de volgende generatie van de hardware zullen we mogelijk ook X-terminals gebruiken.

INFORMATIE JAARGANG 33 NR. 2
PAG. 65 T/M 140 1991



Figuur 1: De architectuur van Amoeba

Ten tweede zijn er de pool-processoren, een verzameling CPU's die zonodig dynamisch kunnen worden gealloceerd, gebruikt en geretourneerd aan de verzameling beschikbare processoren (de 'pool'). Als het *make*-commando bijvoorbeeld zes compilaties zou moeten doen, dan zouden zes processoren uit de pool kunnen worden gehaald gedurende de tijd nodig voor de compilatie en vervolgens kunnen worden geretourneerd. Met een compiler die vijf compilatieslagen nodig heeft, zou een andere mogelijkheid zijn $5 \times 6 = 30$ processoren te alloceren voor de zes compilaties, zodat het proces zelfs nog meer versneld wordt. Veel programmatuur, zoals heuristisch zoeken in toepassingen in de kunstmatige intelligentie (bijvoorbeeld het schaakspel), gebruikt grote aantallen pool-processoren om rekenwerk te verrichten. Op het ogenblik hebben we 48 op VME gebaseerde single-board computers die de 68020 en 68030 CPU's gebruiken. We hebben ook 10 VAX CPU's die een extra processor-pool vormen.
Ten derde zijn er de gespecialiseerde servers, zoals directory-servers, file-servers, gegevensbank-servers, boot-servers en verscheidene andere servers met gespecialiseerde functies. Elke server is er speciaal om een specifieke taak te vervullen. In sommige gevallen zijn er meer servers die dezelfde taak vervullen, bijvoorbeeld als deel van het geduplicеerde filesysteem.
Ten vierde zijn er de gateways, die worden gebruikt om Amoeba-systemen op verschillende lokaties en in verschillende landen aan elkaar te verbinden tot één enkel uniform systeem. De gateways isoleren Amoeba van de eigenaardigheden van de protocollen die moeten worden gebruikt voor de wide-area netwerken.

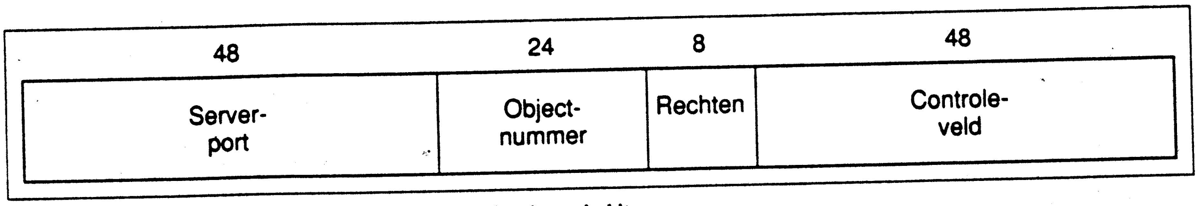
Op alle Amoeba-machines draait dezelfde kernel, die in de eerste plaats zorgdraagt voor processen met vele threads (zie onder), communicatieservice en I/O, en voor weinig anders. De basisgedachte achter de kernel was hem klein te houden, zijn betrouwbaarheid te verhogen, en ervoor te zorgen dat het mogelijk is een zo groot mogelijk deel van het bedrijfssysteem te draaien als een user-proces (een gebruikersproces, dat wil zeggen buiten de kernel), zodat het resultaat flexibel wordt en ruimte laat voor experimenteren.

2.2 Objecten en capabilities

Amoeba is een op objecten gebaseerd systeem. Het systeem kan worden opgevat als een verzameling objecten. Bij elk object hoort een verzameling van operaties die op dat object kan worden uitgevoerd. Bij een file-object bijvoorbeeld zijn kenmerkende operaties: lezen van, schrijven op, toevoegen aan en verwijderen van het object. De lijst van toegestane operaties wordt gedefinieerd door degene die het object ontwerpt en die de code schrijft waarmee het object geïmplementeerd wordt. Er bestaan zowel hardware- als software-objecten.

Elk object gaat vergezeld van een *capability*,[8] een soort pas of sleutel die de eigenaar van de capability de mogelijkheid geeft een aantal (niet noodzakelijkerwijs alle) operaties op dat object uit te voeren. Een gebruikersproces zou bijvoorbeeld een capability voor een file kunnen hebben die hem toestaat van de file te lezen, maar niet de file te veranderen. Capabilities worden cryptografisch beschermd om te voorkomen dat gebruikers eraan kunnen knoeien.
Elk gebruikersproces heeft een zekere verzameling capabilities, die tezamen de verzameling toegankelijke objecten en de daarop uitvoerbare operaties definiëren. Op die manier voorzien capabilities in een algemeen mechanisme voor het benoemen, aanspreken en beschermen van objecten. Vanuit het perspectief van de gebruiker bezien, is de functie van het bedrijfssysteem het creëren van een omgeving waarin objecten op een beschermde manier kunnen worden gecreëerd en gemanipuleerd.

Dit op objecten gebaseerde model, dat zichtbaar is voor de gebruikers, wordt *geïmplementeerd* door middel van 'remote procedure call' (RPC).[5] Elk object gaat vergezeld van een *server*-proces dat de manager is van het object. Als een gebruikersproces een operatie op een object wil uitvoe-



Figuur 2: Een capability. De getallen zijn de huidige afmetingen in bits

ren, zendt het een boodschap met een verzoek naar de server die manager is van het object. Deze boodschap bevat de capability voor het object, een specificatie van de uit te voeren operatie en de eventuele parameters die voor de operatie nodig zijn. De gebruiker, die *cliënt* wordt genoemd, wordt dan geblokkeerd. Als de server de operatie heeft uitgevoerd, stuurt hij een antwoordboodschap terug, waardoor de blokkade van de cliënt wordt opgeheven. De combinatie van het sturen van een boodschap met een verzoek, het blokkeren, en het ontvangen van een antwoordboodschap, vormt de remote procedure call. Om de gehele operatie op afstand eruit te laten zien als een aanroep van een lokale procedure, kan de remote procedure call worden omvat in zogenaamde stub-routines (hoewel: zie Tanenbaum en Van Renesse, 1988[27]).

De structuur van een capability wordt getoond in figuur 2. Een capability is 128 bits lang en bevat vier velden. Het eerste veld is de *server-port* en wordt gebruikt om het (server-)proces te identificeren dat de manager is van het object. In feite is het een willekeurig getal van 48 bits, dat wordt gekozen door de server.

Het tweede veld is het *objectnummer* dat wordt gebruikt door de server om uit te vinden welk van zijn objecten wordt aangesproken. Samen identificeren de server-port en het objectnummer op een unieke manier het object waarop de operatie moet worden uitgevoerd.
Het derde veld is het *rechten*veld, dat een bitmap bevat die vertelt welke operaties de eigenaar van de capability mag uitvoeren. Als alle bits 1 zijn, zijn alle operaties toegestaan. Als daarentegen een aantal bits 0 is, mag de eigenaar van de capability de corresponderende operaties niet uitvoeren. Omdat het aantal operaties gewoonlijk beperkt is, is 8 bits voldoende.

Om te voorkomen dat gebruikers gewoon alle 0-bits in het rechtenveld omzetten in 1-bits, wordt een cryptografisch beschermingssysteem gebruikt. Als een server wordt gevraagd een object te creëren, kiest hij een beschikbare lokatie in zijn interne tabellen en zet daar de informatie over het object neer, samen met een vers gegenereerd willekeurig getal van 48 bits. De index in de tabel wordt in het objectnummerveld van de capability gezet, de rechten-bits worden allemaal op 1 gezet en het vers gegenereerde willekeurige getal wordt in het *controleveld* van de capability gezet. Dit is een zogenaamde *owner capability*, die kan worden gebruikt om alle operaties op het object uit te voeren.

De eigenaar kan een nieuwe capability met een deelverzameling van de mogelijke rechten construeren door een aantal rechten-bits uit te zetten en vervolgens het rechtenveld te XOR-en met het willekeurige getal in het controleveld. Het resultaat van deze operatie wordt dan door een *éénrichtingsfunctie* gevoerd, hetgeen een nieuw getal van 48 bits oplevert dat in het controleveld van de nieuwe capability gezet wordt.

De sleuteleigenschap die wordt vereist van de éénrichtingsfunctie, f, is dat het, gegeven het originele 48-bits getal N en het ongecodeerde rechtenveld R, eenvoudig is om $C = f(N \text{ XOR } R)$ te berekenen, maar dat het gegeven alleen C nagenoeg onmogelijk is een argument voor f te vinden dat de gegeven C oplevert. Zulke functies zijn bekend.[9]
Als een capability bij een server aankomt, gebruikt de server het objectveld als index in zijn tabellen om de informatie over het object op te zoeken. Dan gaat hij na of alle rechten-bits aanstaan. Is dat het geval, dan weet de server dat de capability een owner-capability is (of lijkt te zijn) en dus vergelijkt hij gewoon het oorspronkelijke willekeurige getal met de inhoud van het controleveld. Komen ze overeen, dan wordt de capability geldig geacht en wordt de gewenste operatie uitgevoerd.
Als een aantal rechten-bits 0 is, weet de server dat hij te maken heeft met een afgeleide capability, dus voert hij een XOR uit van het oorspronkelijke willekeurige getal in zijn tabel met het rechtenveld van de capability. Dit getal wordt dan door de éénrichtingsfunctie gevoerd. Als de uitvoer van de éénrichtingsfunctie overeenkomt met de inhoud van het controleveld, dan wordt de capability geldig geacht en wordt de gevraagde operatie uitgevoerd mits het corresponderende rechten-bit op 1 staat. Als gevolg van het feit dat de éénrichtingsfunctie niet inverteerbaar is, is het voor de gebruiker niet mogelijk een capability te 'ontcijferen' om het oorspronkelijke willekeurige getal te vinden met als doel daarmee een vervalste capability met meer rechten te creëren.

2.3 Remote operations

De combinatie van een verzoek van een cliënt aan een server en een antwoord van een server aan een cliënt heet een *remote operation* (een operatie op afstand). De vraag- en antwoordboodschappen bestaan uit een header en een buffer. Headers zijn 32 bytes groot en buffers kunnen tot 30 kilobytes groot zijn. Een header voor een verzoek bevat de capability van het object waarop de operatie moet worden uitgevoerd, de operatiecode en een beperkte ruimte (8 bytes) voor parameters voor de operatie. Bij een schrijfoperatie op een file bijvoorbeeld, identificeert de capability de file, is de operatiecode *write* en specificeren de parameters de hoeveelheid te schrijven data en het beginpunt in de file. De buffer voor zo'n verzoek bevat de te schrijven gegevens. Een antwoord-header bevat een foutcode, een beperkte ruimte voor het resultaat van de operatie (8 bytes) en een capability-veld dat kan worden gebruikt om een capability te retourneren (bijvoorbeeld als resultaat

van de creatie van een object of van een zoekoperatie in directories).

De elementaire operaties voor het uitvoeren van remote operations zijn:

get_request(req-header, req-buffer, req-grootte)
put_reply(rep-header, rep-buffer, rep-grootte)
do_operation(req-header, req-buffer, req-grootte, rep-header, rep-buffer, rep-grootte)

Als een server gereed is om verzoeken van cliënten te accepteren, voert hij de elementaire operatie *get-request* uit, waardoor hij wordt geblokkeerd. Als een boodschap met een verzoek arriveert, wordt de blokkade van de server opgeheven en worden de formele parameters van de aanroep van *get-request* gevuld met de informatie van het binnenkomende verzoek. De server voert vervolgens het werk uit en stuurt een antwoord met *put-reply*.

Aan de kant van de cliënt gebruikt een proces *do-operation* om een remote operation aan te roepen. Deze handeling zorgt ervoor dat de boodschap met het verzoek wordt gezonden aan de server. De header van het verzoek bevat de capability van het te manipuleren object en verscheidene parameters die betrekking hebben op de operatie. Het proces wordt geblokkeerd tot het antwoord is ontvangen; als dat is gebeurd, worden de drie rep-parameters ingevuld en wordt een status geretourneerd. De teruggegeven status van *do-operation* heeft drie mogelijke waarden:

1 Het verzoek is aangekomen en is uitgevoerd.
2 Het verzoek is niet aangekomen of niet uitgevoerd (bijvoorbeeld omdat de server down was).
3 De status is onbekend.

Het derde geval kan optreden als een verzoek is verzonden (en mogelijkerwijs bevestigd), maar geen antwoord wordt ontvangen. Deze situatie doet zich voor als een server halverwege de remote operation uitvalt. Onder alle omstandigheden van verdwenen boodschappen en uitgevallen servers garandeert Amoeba dat boodschappen hoogstens één keer worden bezorgd. Als status 3 wordt geretourneerd, wordt het aan de applicatie of het run-time systeem overgelaten om de fout op te vangen.

2.4 Aanroepen van remote procedures

Een aanroep van een remote procedure bestaat in feite uit meer dan alleen de vraag/antwoord-uitwisseling die boven werd beschreven. De cliënt moet de capability, de operatie-code en de parameters in de buffer van het verzoek stoppen en als het antwoord binnen is, moeten de resultaten worden uitgepakt. De server moet de capability controleren, de operatiecodes en de parameters uit het verzoek halen en de passende procedure aanroepen. Het resultaat van de procedure moet in de antwoordbuffer worden gezet. Het plaatsen van parameters of resultaten in een buffer van een boodschap heet *marshalling* en de kosten daarvan zijn niet-triviaal. Ook moet rekening worden gehouden met verschillen in gegevensrepresentatie tussen cliënt en server. Al deze stappen moeten voorzichtig worden ontworpen en gecodeerd, opdat ze geen onacceptabele overhead introduceren.

Om het marshalling en het sturen van boodschappen voor de gebruikers te verbergen gebruikt Amoeba *stub-routines*.[5] Eén van de filesysteem-stubs zou bijvoorbeeld kunnen beginnen met:

```
int lees_file(file_cap, beginpunt, nbytes, buffer,
bytes_gelezen)
capability_t*file_cap;
long beginpunt;
long*nbytes;
char*buffer;
long*bytes_gelezen;
```

Deze routine leest *nbytes* beginnend bij *beginpunt* van de file die wordt geïdentificeerd door *file_cap* en zet het resultaat in *buffer*. In *bytes_gelezen* wordt het in werkelijkheid gelezen aantal bytes geretourneerd. De functie zelf retourneert 0 als de executie correct verloopt en een foutcode in de overige gevallen. Een met de hand geschreven stub is voor deze code eenvoudig te construeren: de stub zal een header voor een vraagboodschap produceren waar *file_cap* in staat, de operatiecode voor *lees_file*, *beginpunt* en *nbytes* en vervolgens de volgende remote operation aanroepen:

```
do_operation(req_hdr, req_buf, req_bytes, rep_hdr,
buf, rep_bytes);
```

Het is onmogelijk zo'n stub automatisch te genereren uit bovenstaande procedure-header. Er ontbreekt een hoeveelheid essentiële informatie. De schrijver van de handgeschreven stub gebruikt een zekere hoeveelheid afgeleide informatie om het werk te doen.

1 De buffer wordt alleen gebruikt om informatie van de file-server te ontvangen; de buffer is een uitvoerparameter en moet niet naar de server gezonden worden.
2 De maximumlengte van de buffer wordt gegeven in de *nbytes* parameter. De werkelijke lengte van de buffer is de geretourneerde waarde als er geen fout is, en anders nul.
3 *File_cap* is speciaal; het definieert de service die de remote operation moet verlenen.
4 De stub-generator weet niet wat de operatiecode van

de server voor *lees file* is. Dit vergt extra informatie. Maar, om eerlijk te zijn, de menselijke stub-schrijver heeft deze extra informatie ook nodig.

Om automatische stub-generatie mogelijk te maken moeten de interfaces tussen cliënt en servers de bovengenoemde informatie bevatten, plus informatie omtrent typerepresentaties voor alle taal/machine-combinaties die in gebruik zijn. Bovendien moeten de interface-specificaties een *overervingsmechanisme* hebben dat het mogelijk maakt dat interfaces van een lager niveau worden gedeeld door verschillende andere interfaces. De *lees file*-operatie bijvoorbeeld zal worden gedefinieerd in een interface op een laag niveau die dan overgeërfd wordt door alle file-server-interfaces, de terminal-server-interface en de segment-server-interface.

AIL (Amoeba Interface Language) is een taal waarin de extra informatie kan worden gespecificeerd voor de generatie van efficiënte stubs, zodat de AIL-compiler automatisch stub-routines kan produceren.[33] De *lees file*-operatie zou onderdeel van een interface (in AIL *class* genaamd) kunnen zijn waarvan de definitie er ongeveer als volgt uit zou kunnen zien:

```
class simpele file server [1000..1999]{
lees file(*, in unsigned beginpunt, in out unsigned
nbytes, out char buffer nbytes: NBYTES;)
schrijf file(*, . . . );
};
```

Uit deze specificatie kan AIL de stub van de cliënt uit het bovengenoemde voorbeeld genereren met de correcte code voor marshalling. AIL kan ook de hoofdroutine van de server genereren die de marshalling code bevat die correspondeert met de stubs van de cliënt. De AIL-specificatie vertelt AIL dat de operatiecodes voor de simpele file server gealloceerd kunnen worden in het bereik van 1000 tot 1999; de specificatie vertelt welke parameters invoer- en welke parameters uitvoerparameters voor de server zijn en hij vertelt dat de lengte van de buffer hoogstens *NBYTES* is (wat een constante moet zijn) en dat de werkelijke lengte *nbytes* is.
De Bullet File Server, één van de file-servers die in Amoeba operationeel zijn, *overerft* deze interface, zodat hij deel gaat uitmaken van de Bullet File Server interface:

```
class bullet server [2000..2999]{
inherit simpele file server;
create file(*, . . . );
};
```

AIL ondersteunt *meervoudige overerving*, dus de Bullet File Server kan zowel de eenvoudige file interface erven als, bijvoorbeeld, een *capability management* interface voor het beperken van rechten op capabilities.
Op het ogenblik genereert AIL stubs in C, maar Modula-stubs en stubs in andere talen zijn gepland. AIL stubs zijn ontworpen voor het omgaan met verschillen in gegevensrepresentatie – zoals de volgorde van bytes en representatie van getallen in drijvende-kommanotatie – op cliënt- en server-machines.

2.5 Threads

Een proces in Amoeba bestaat uit één of meer *threads* die parallel draaien. Alle threads van een proces delen dezelfde adresseringsruimte, maar elke afzonderlijke thread heeft een eigen gedeelte van die adresseringsruimte in gebruik voor zijn persoonlijke stack en heeft zijn eigen programmateller. Vanuit het gezichtspunt van de programmeur lijkt elke thread op een traditioneel sequentieel proces, behalve dat de threads van een proces met elkaar kunnen communiceren via hun gemeenschappelijke geheugen. Daarnaast kunnen de threads (optioneel) onderlinge synchronisatie bewerkstelligen door gebruik te maken van mutexes of semaforen.
Het optreden van meer threads in één proces is bedoeld om de prestaties te verbeteren: nu kan immers gebruik worden gemaakt van parallelle processen en wordt tegelijkertijd een redelijk semantisch model aan de programmeur aangeboden. Een file-server bijvoorbeeld zou kunnen worden geprogrammeerd als een proces met meer threads. Als een verzoek binnenkomt, kan het aan een zekere thread gegeven worden, die het in behandeling neemt. Die thread controleert eerst een interne (software-)cache om te zien of de benodigde gegevens aanwezig zijn. Is dat niet het geval, dan voert hij een RPC uit met een remote disk-server om de gegevens te verkrijgen.
Tijdens het wachten op het antwoord van de disk is de thread geblokkeerd en niet in staat andere verzoeken in behandeling te nemen. Nieuwe verzoeken kunnen echter aan andere threads in hetzelfde proces gegeven worden, zodat eraan gewerkt kan worden terwijl de eerste thread geblokkeerd is. Op deze manier kunnen verschillende verzoeken tegelijkertijd behandeld worden, terwijl elke thread zijn werk op een sequentiële manier kan doen. Het nut van het feit dat alle threads een gemeenschappelijke adresseringsruimte hebben, is dat het mogelijk gemaakt wordt dat ze allemaal toegang hebben tot een gemeenschappelijke cache. Dat is niet mogelijk als elke thread zijn eigen adresseringsruimte heeft.

Het coördineren van threads binnen een proces wordt gedaan door de code in het proces zelf. Als een thread niet verder kan, òf omdat hij geen werk te doen heeft (d.w.z. ten gevolge van een *get request*), òf omdat hij wacht op een remote antwoord (d.w.z. ten gevolge van een *do operation*), wordt de interne coördinator aangeroepen en de

thread geblokkeerd, en kan een nieuwe thread draaien. In feite zijn threads dus co-routines. Threads worden niet voortijdig onderbroken, wat wil zeggen dat de thread die op het ogenblik draait, niet tot stilstand zal worden gebracht omdat hij al te lang gedraaid heeft. Dit besluit werd genomen om race-condities te vermijden. Een thread hoeft zich geen zorgen te maken dat hij, terwijl hij halverwege het veranderen van een kritieke gedeelde tabel is, wordt gestopt en dat een andere thread wordt opgestart die probeert dezelfde tabel te gebruiken.

Aangenomen wordt dat de threads in een proces alle door dezelfde programmeur geschreven worden en actief samenwerken. Daarom zitten ze in hetzelfde proces. De interactie tussen twee threads in hetzelfde proces is dus nogal verschillend van de interactie tussen twee threads van verschillende processen, die elkaar vijandig gezind kunnen zijn en waarvoor geheugenbescherming in hardware noodzakelijk en aanwezig is. Onze evaluatie van deze aanpak wordt later besproken.

3 Servers

De Amoeba kernel, die boven werd beschreven, verzorgt in essentie de communicatie en het procesmanagement en doet verder weinig. De kernel zorgt voor het versturen en ontvangen van boodschappen, het coördineren van processen en wat memory-management op een laag niveau. Al het andere werk wordt gedaan door gebruikersprocessen. Zelfs capability-management wordt in zijn geheel buiten de kernel uitgevoerd, omdat de cryptografische techniek die eerder werd beschreven, het voor gebruikers haast onmogelijk maakt gefalsificeerde capabilities te genereren.

Alle resterende functies die normaal gesproken worden geassocieerd met een modern bedrijfssysteem, worden uitgevoerd door servers, die gewoon normale gebruikersprocessen zijn. Het filesysteem, bijvoorbeeld, bestaat uit een verzameling gebruikersprocessen. Gebruikers die niet tevreden zijn met het standaard-filesysteem zijn vrij om hun eigen systeem te schrijven en dat te gebruiken. Deze situatie staat in contrast met een systeem als UNIX, waarin er maar één enkel filesysteem is dat alle applicaties moeten gebruiken, ook al is het nog zo weinig passend. Stonebraker[24] beschrijft bijvoorbeeld de talrijke problemen die UNIX schept voor gegevensbanksystemen.

In de volgende paragrafen zullen we als voorbeelden van typerende Amoeba-servers achtereenvolgens de Amoeba geheugen-server, de proces-server, de file-server en de directory-server beschrijven. Er bestaan nog vele andere.

3.1 De geheugen-server en proces-server

In veel applicaties hebben processen een methode nodig om subprocessen te creëren. In UNIX wordt een subproces gecreëerd door de elementaire operatie *fork*, waarin een exacte kopie wordt gemaakt van het originele proces. Dit proces kan dan een tijdje draaien terwijl het zich bezighoud met huishoudelijke activiteiten en dan de elementaire operatie *exec* uitvoeren om zijn core image te overschrijven met een nieuw programma.

In een gedistribueerd systeem is dit model niet aantrekkelijk. Het idee eerst een exacte kopie van het proces te maken, mogelijk remote, en die kort daarna weer weg te gooien, is inefficiënt. Als gevolg hiervan past Amoeba een andere strategie toe. De sleutelconcepten zijn segment- en procesdescriptoren, die hieronder worden beschreven.

Een *segment* is een aaneengesloten stuk geheugen dat code of gegevens kan bevatten. Elk segment heeft een capability die de houder ervan toestaat er operaties op uit te voeren zoals lezen en schrijven. Een segment is ongeveer een in-core file met vergelijkbare eigenschappen.

Een *procesdescriptor* is een datastructuur die informatie levert over een *stunned* (letterlijk 'bedwelmd') proces, dat wil zeggen een proces dat nog niet gestart is of dat wordt gedebugged of gemigreerd. Een procesdescriptor heeft vier componenten. De eerste beschrijft de benodigdheden voor het systeem waarop het proces moet draaien: de klasse van machines, welke instructieset, het minimum aan beschikbaar geheugen, het gebruik van speciale instructies zoals drijvende-komma-instructies, en verscheidene andere benodigdheden. De tweede component beschrijft de lay-out van de adresruimte: het aantal segmenten en voor elk segment de grootte, het virtueel adres, hoe het is afgebeeld (bijvoorbeeld alleen leesbaar, leesbaar/schrijfbaar, code/data-ruimte) en de capability van een file of segment die de data van het segment bevat. De derde component beschrijft de toestand van elke thread: stackpointer, stacktop en -bodem, programmateller, processor-statuswoord en registers. Threads kunnen zijn geblokkeerd door sommige system-calls (bijvoorbeeld get request); dit kan ook worden beschreven. De vierde component is een lijst van ports waarvoor het proces een server is. Deze lijst is van nut voor de kernel als binnenkomende verzoeken moeten worden gebufferd en antwoord moet worden gegeven aan port-locate-operaties (die hieronder beschreven zullen worden).

Een proces wordt gecreëerd door de volgende stappen uit te voeren:

1 Haal de procesdescriptor voor de binaire programmacode uit het filesysteem.
2 Creëer een lokaal segment of een file en initialiseer deze met de beginomgeving van het nieuwe proces. De omgeving bestaat uit een verzameling benoemde capabilities (een primitieve directory als het ware) en de argumenten voor het proces (in UNIX-termen: *argc* en *argv*).
3 Verander de procesdescriptor zo dat het zojuist gecreëerde omgevingssegment het eerste segment wordt.
4 Zend de procesdescriptor naar de machine waar hij zal worden uitgevoerd.

Als de procesdescriptor aankomt bij de machine waar het proces zal draaien, extraheert de geheugen-server daar de capabilities voor de remote segmenten, en haalt hij de code- en datasegmenten van waar ze zich maar bevinden met behulp van de capabilities voor het uitvoeren van read-operaties. Op deze manier zijn de fysieke lokaties van alle betrokken machines irrelevant.
Als eenmaal alle segmenten zijn ingevuld, kan het proces worden geconstrueerd en worden gestart. Een capability voor het proces wordt geretourneerd aan degene die het proces startte. Deze capability kan worden gebruikt om het proces te beëindigen, of het kan aan een debugger gegeven worden om het *stunned* te maken (tijdelijk te stoppen), zijn geheugen te lezen en te schrijven, enzovoort.

3.2 De file-server

Wat het systeem betreft is een file-server gewoon een gebruikersproces. Bijgevolg zijn voor Amoeba in de loop van zijn bestaan verscheidene file-servers geschreven. De eerste, *FUSS (Free University Storage System)*,[17] werd ontworpen als een experiment met het management van concurrent access met gebruik van optimistische concurrency-controle. De huidige file-server, de *Bullet Server*, werd ontworpen met extreem hoge prestaties als doel.[29,31,32] Deze file-server zullen we hieronder beschrijven.

De prijsdaling van diskgeheugen en RAM in de laatste tien jaar maakte het ons mogelijk een radicaal ander ontwerp te gebruiken dan wat gebruikt werd in UNIX en de meeste andere bedrijfssystemen. We hebben in het bijzonder het idee verlaten om files op te slaan als een verzameling disk-blokken van vaste grootte. Alle files worden aaneengesloten opgeslagen, zowel op de disk als in het hoofdgeheugen van de server. Hoewel dit ontwerp wat schijfruimte en geheugen verspilt door de overhead ten gevolge van fragmentatie, vinden we dat de enorme prestatiewinst (die hieronder zal worden beschreven) de lage extra kosten van de aanschaf van, zeg, een 800 MB disk in plaats van een 500 MB disk om 500 MB aan files op te slaan, ruimschoots compenseert.

De Bullet Server is een opslagplaats voor onveranderbare files, met als hoofdoperaties *read-file* en *create-file*. (Er is ook een *delete-file*-operatie voor gebruik bij garbage-collection). Als een proces een *read-file*-verzoek doet, kan de Bullet Server de hele file naar de cliënt overbrengen in een enkele RPC, tenzij hij groter is dan de maximumgrootte (30.000 bytes). In het laatste geval zijn meer RPC's nodig. De cliënt kan dan editen of op een andere manier de file lokaal modificeren. Als de cliënt klaar is, voert hij een *create-file* RPC uit om een nieuwe versie te maken. De oude versie blijft intact totdat hij expliciet wordt verwijderd of door garbage-collection wordt weggegooid. Merk op dat verschillende versies van een file verschillende capabilities hebben, zodat ze naast elkaar kunnen bestaan. Dit maakt het gemakkelijk source-code controlesystemen te implementeren.
De files worden aaneengesloten op disk opgeslagen en worden gecached in het geheugen van de file-server (op het ogenblik 12 Mbytes). Als een gevraagde file niet in dit geheugen beschikbaar is, dan wordt ze van de disk gehaald in één enkele grote DMA-operatie en aaneengesloten opgeslagen in de cache. (In tegenstelling tot conventionele filesystemen worden nergens in het filesysteem 'blokken' gebruikt.) In de *create-file*-operatie kan men antwoord eisen voordat de file op de disk is geschreven (voor de snelheid), of daarna (om zeker te weten dat de file succesvol is weggeschreven).
Als de Bullet Server wordt opgestart, wordt de gehele 'i-node tabel' in het geheugen gelezen in één enkele disk-operatie en deze tabel blijft daar zolang de server draait. Als een file-operatie wordt gevraagd, wordt uit de capability het objectnummerveld gehaald, dat een index is in deze tabel. Het tabelelement dat zo is gelokaliseerd, geeft zowel het disk-adres als het cache-adres van de aaneengesloten file (als dat aanwezig is). Er is geen disk-operatie nodig om de 'i-node' op te halen en er is hoogstens één disk-operatie nodig om de file zelf op te halen, als deze zich niet in de cache bevindt. De eenvoud van dit ontwerp ruilt wat ruimte in tegen goede prestaties.

3.3 De directory-server

De Bullet Server zorgt niet voor het benoemen van files. Om toegang te krijgen tot een file moet een proces de relevante capability leveren. Omdat het voor mensen niet prettig is te werken met 128-bits binaire getallen hebben we een directory-server ontworpen en geïmplementeerd om namen en directories bij te houden. De directory-server houdt verschillende directories bij. Elke directory is een normaal object. In essentie geeft een directory een afbeelding van ASCII strings naar capabilities. Een proces kan een string, bijvoorbeeld een filenaam, geven aan de directory-server en de directory-server geeft de capability voor die file terug. Door deze capability te gebruiken kan het proces dan toegang krijgen tot de file.

Als een file wordt geopend, in UNIX-termen, levert de directory-server de capability voor het gebruik bij lees- en schrijfoperaties. Nadat de capability bij de directory-server is verkregen, gaan de daarop volgende RPC's direct naar de server die manager is van het betreffende object. De directory-server is er dan niet langer bij betrokken.
Het is belangrijk zich te realiseren dat de directory-server simpelweg een afbeeldingsfunctie voorstelt. De cliënt levert een capability voor een directory (om aan te geven in welke directory moet worden gezocht) en een string, en de directory-server zoekt de string op in de gespecificeerde

directory en retourneert de capability die is geassocieerd met de string. De directory-server bezit geen kennis van het soort object dat de capability controleert.
Het kan bijvoorbeeld een capability zijn voor een andere directory op dezelfde of een andere directory-server, een file, een mailbox, een gegevensbank, een proces-capability, een segment-capability, een capability voor een stuk hardware, of nog iets anders. Bovendien kan de capability horen bij een object op dezelfde machine, een object op een andere machine op een lokaal netwerk, of een object in een ander land. De aard en de lokatie van het object zijn volledig arbitrair. De objecten in een directory hoeven dus niet allemaal op dezelfde disk te staan, zoals bijvoorbeeld het geval is bij veel systemen die 'remote mount'-operaties ondersteunen.
Doordat een directory capabilities voor andere directories mag bevatten, is het mogelijk arbitraire directory-structuren te bouwen, waaronder bomen en grafen. Als een optimalisatie is het mogelijk aan de directory-server een compleet pad op te geven en de server dat pad zo ver als hij kan te laten volgen, waarna hij uiteindelijk een enkele capability retourneert.

Eigenlijk zijn directories iets algemener dan gewoon eenvoudige afbeeldingen. Het is gewoonlijk het geval dat de eigenaar van een file het recht wil hebben alle operaties op de file uit te voeren, maar anderen alleen read-only-toegang wil geven. De directory-server ondersteunt dit idee door directories te structureren als een serie rijen, één per object, zoals wordt getoond in figuur 3.

De eerste kolom levert de string (bijvoorbeeld de filenaam). De tweede kolom geeft de capability die bij die string hoort. De overige kolommen hebben elk betrekking op een klasse van gebruikers. Men zou bijvoorbeeld een directory kunnen opzetten met verschillende toegangsrechten voor de eigenaar, de groep van de eigenaar en anderen, zoals in UNIX, maar andere combinaties zijn ook mogelijk.
De capability voor een directory specificeert de kolommen waartoe de houder toegang heeft in de vorm van een bitmap in een gedeelte van het rechtenveld (bijvoorbeeld 3 bits). In het bovenstaande voorbeeld zouden de bits 001 toegang kunnen omschrijven tot de *Anderen*-kolom en geen van de andere kolommen. Eerder bespraken we hoe de rechten-bits worden beschermd tegen onrechtmatig gebruik door middel van het controleveld.

objectnaam	*capability*	*eigenaar*	*groep*	*anderen*
	cap1	11111	11000	10000
games-dir	cap2	11111	10000	10000
artikel.t	cap3	11111	00000	00000
prog.c	cap4	11111	11100	10000

Figuur 3: Een directory met drie gebruikersklassen, vier elementen en vijf rechten

Beschouw een typerend toegangsverzoek om te zien hoe men meer dan één kolom kan gebruiken. De cliënt levert een capability voor een directory (wat een kolom impliceert) en een string. De string wordt opgezocht in de directory om de bijbehorende rij te vinden. Vervolgens wordt de kolom vergeleken met de (singleton) bitmap in het rechtenveld, om te zien welke kolom moet worden gebruikt. Ter herinnering: het eerder beschreven cryptografische systeem maakt het onmogelijk dat gebruikers de bitmap wijzigen en daardoor toegang verkrijgen tot een verboden kolom.
Dan wordt het element in de geselecteerde rij en kolom genomen. Conceptueel is dit gewoon een capability, met de passende rechten-bits aangezet. Om echter niet heel veel capabilities, waarvan er weinig ooit worden gebruikt, op te hoeven slaan, wordt een optimalisatie toegepast en is de entry gewoon een bitmap, *b*. De directory-server kan vervolgens aan de server die manager is van het object, vragen een nieuwe capability te retourneren met alleen die rechten die in *b* zitten. Deze nieuwe capability wordt geretourneerd aan de gebruiker en tevens gecached voor gebruik in de toekomst om het aantal aanroepen van de server te beperken.

De directory-server ondersteunt een aantal operaties op directory-objecten. Dat zijn onder andere het opzoeken van capabilities, het toevoegen van nieuwe rijen aan een directory, het verwijderen van rijen uit directories, het opsommen van de inhoud van directories, het informeren naar de status van directories en objecten en het verwijderen van directories. Er is ook een voorziening voor het uitvoeren van meer operaties in één enkele atomaire actie om de fouttolerantie te verhogen.
Bovendien bestaat er ook ondersteuning voor het hanteren van geduplіceerde objecten. Het capability-veld in figuur 3 kan in feite een verzameling capabilities bevatten voor diverse kopieën van elk object. Op die manier kan een proces dat een object opzoekt, de volledige verzameling capabilities voor alle kopieën ophalen. Als één van de objecten niet beschikbaar is, kunnen de andere worden geprobeerd. De techniek lijkt op die, die Eden gebruikte.[20]
Daarnaast is er een optie beschikbaar waarbij de directory-server zelf een verzoek doet voor het maken van kopieën als een nieuw object wordt geïnstalleerd in een directory; vervolgens slaat hij alle bijbehorende capabilities op. Op die manier wordt de gebruiker van dit administratieve werk bevrijd.

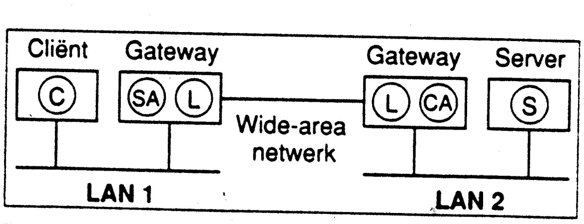
Naast de ondersteuning van duplicatie van objecten wordt de directory-server zelf geduplіceerd. Daardoor is het on-

der andere mogelijk nieuwe versies van de server te installeren door één van de instanties ervan te vernietigen, een nieuwe versie te installeren als vervanging, de andere (originele) instantie te vernietigen en de tweede vervanging, die óók de nieuwe code draait, te installeren. Op deze manier kunnen bugs worden gerepareerd zonder dat de service onderbroken wordt.

4 Wide-area Amoeba

Amoeba werd ontworpen met het idee dat een verzameling machines op een LAN de mogelijkheid zou hebben over een wide-area netwerk te communiceren met een vergelijkbare verzameling remote machines. Het hoofdprobleem is hier dat wide-area netwerken langzaam en onbetrouwbaar zijn en daarnaast protocols als X.25, TCP/IP en OSI gebruiken; in ieder geval, geen RPC. Het hoofddoel van de omgang met wide-area netwerken in Amoeba was transparantie te bereiken zonder prestatie in te hoeven leveren.[30] Het is in het bijzonder niet wenselijk dat de snelle lokale RPC wordt vertraagd als gevolg van het bestaan van wide-area communicatie. We geloven dat dit doel is bereikt.

De Amoeba-wereld is onderverdeeld in *domeinen*, waarbij elk domein een onderling verbonden verzameling van local-area netwerken is. Het hoofdaspect van een domein (bijvoorbeeld een campus) is dat broadcasts (uitzendingen) van een willekeurige machine in het domein worden ontvangen door alle andere machines in het domein, maar niet door machines buiten het domein.
Het belang van broadcasting hangt samen met de wijze waarop ports worden gevonden in Amoeba. Als een proces een RPC doet met een port die nog niet eerder werd gebruikt, broadcast de kernel een *locate*-boodschap. De server antwoordt op deze broadcast met zijn adres, dat dan wordt gebruikt en tevens wordt gecached voor toekomstige RPC's.
Deze strategie is niet wenselijk in een wide-area netwerk. Hoewel broadcasting kan worden gesimuleerd door gebruik te maken van een *minimale opspannende boom*,[7] is het duur en inefficiënt. Bovendien hoeft niet elke service wereldwijd beschikbaar te zijn. Een laserprinter-service in het natuurkundegebouw van een universiteit in Californië zal bijvoorbeeld niet van veel nut zijn voor cliënten in New York.



Figuur 4: Bij wide-area communicatie in Amoeba zijn zes processen betrokken

Deze beide problemen worden aangepakt door de introductie van het concept *publikatie*. Als een service buiten zijn eigen domein bekend en toegankelijk wil zijn, neemt hij contact op met de *Service for Wide-Area Networks* (SWAN) en verzoekt deze zijn port te publiceren in een zekere verzameling domeinen. De SWAN publiceert de port door RPC's te doen met SWAN-processen in elk van de andere domeinen.
Als een port wordt gepubliceerd in een domein, wordt in dat domein een nieuw proces gecreëerd, genaamd de *server-agent*. Het proces draait gewoonlijk op de gateway machine en doet een *get-request* met de port van de remote server. Hij is stil totdat zijn server nodig is; op dat moment komt hij tot leven en doet een RPC met de server.

Laten we nu eens bekijken wat er gebeurt als een proces een remote server wiens port is gepubliceerd, probeert te lokaliseren. De kernel van het proces broadcast een *locate* die wordt ontvangen door de server-agent. De server-agent construeert dan een boodschap en geeft deze aan een *link*-proces op de gateway-machine. Het link-proces stuurt de boodschap door over het wide-area netwerk naar het domein van de server, waar hij bij de gateway aankomt, waardoor een *cliënt-agent*-proces wordt gecreëerd. Deze cliënt-agent doet dan een gewone RPC met de server. De verzameling betrokken processen wordt getoond in figuur 4.

Het mooie van deze opzet is dat het volledig transparant plaatsvindt. Gebruikersprocessen noch de kernel weten welke processen lokaal zijn en welke remote. De communicatie tussen de cliënt en de server-agent is volledig lokaal en voltrekt zich via normale RPC. Evenzo verloopt de communicatie tussen de cliënt-agent en de server volledig normaal. De cliënt noch de server weet dat hij praat met een ergens anders gesitueerd proces.
Natuurlijk zijn de twee agenten zich wel bewust van wat er gebeurt, maar zij worden automatisch gegenereerd als ze nodig zijn en zijn niet zichtbaar voor de gebruikers. De linkprocessen zijn de enige die de details van het wide-area netwerk kennen. Ze praten met de agenten door middel van RPC, maar met elkaar door middel van het protocol dat het wide-area netwerk voorschrijft. Het nut van het afsplitsen van de agenten van de linkprocessen is het volledig isoleren van de technische details van het wide-area netwerk in één soort proces, en het maakt het eenvoudiger om multiway-gateways te gebruiken, die één type linkproces zouden hebben voor elk wide-area netwerktype waar de gateway aan vastzit.

Het is belangrijk op te merken dat dit ontwerp geen significante prestatievermindering oplevert voor lokale communicatie. Een RPC tussen een cliënt en een server op hetzelfde LAN voltrekt zich op volle snelheid, zonder dat er

gerelayeerd hoeft te worden. Er is natuurlijk wat prestatieverlies als een cliënt praat met een server op een ergens anders gelegen netwerk, maar de beperkende factor is normaal gesproken de bandbreedte van het wide-area netwerk, dus de extra overhead als gevolg van het herhaaldelijk relayeren van boodschappen is verwaarloosbaar.

Nog een nuttig aspect van dit ontwerp is de factor management. Om te beginnen kunnen services alleen worden gepubliceerd met behulp van de SWAN-server, die kan nagaan of het systeembeheer de port wel gepubliceerd wil hebben. Een andere belangrijke controlefactor is de mogelijkheid te voorkomen dat zekere processen (bijvoorbeeld die van studenten) toegang hebben tot wide-area services, omdat dit verkeer de gateways moet passeren en daar verscheidene checks kunnen worden uitgevoerd. Tot slot kunnen de gateways financiële administratie doen, statistische gegevens verzamelen en het wide-area netwerk in de gaten houden.

5 Applicaties

Amoeba is gebruikt voor het programmeren van verscheidene applicaties. In deze paragraaf zullen we er een aantal van beschrijven, waaronder UNIX-emulatie, een parallelle versie van *make*, het handelsreizigersprobleem en de alfa/bèta-zoekmethode.

5.1 UNIX-emulatie

Een van de doelen van Amoeba was het systeem bruikbaar te maken als een omgeving voor de ontwikkeling van programmatuur. Voor zo'n omgeving heb je editors, compilers en veel andere standaardsoftware nodig. Besloten werd dat de eenvoudigste manier om deze software te verkrijgen was, UNIX te emuleren en vervolgens daarbovenop UNIX en MINIX[25] compilers en andere hulpprogrammatuur te draaien.
Door gebruik te maken van een speciale verzameling bibliotheekroutines die RPC's doen met de Amoeba-servers, was het mogelijk een emulatie van de UNIX systemcall interface te construeren – die de naam *Ajax* kreeg – die in ieder geval zó goed is dat ongeveer 100 van de meest voorkomende utility-programma's (hulpprogrammatuur) konden worden overgezet naar Amoeba. De Amoeba-gebruiker kan nu de meeste standaard-editors, compilers, file-utilities en andere programma's gebruiken op een manier die erg lijkt op UNIX, hoewel het in feite Amoeba is. Er is gezorgd voor een *session-server* om toestandsinformatie te verwerken en *fork* en *exec* op een UNIX-achtige manier uit te voeren.

5.2 Een parallelle versie van *make*

Zoals te zien is in figuur 1, bevat de hardware waarop Amoeba draait een processor-pool met een paar dozijn 68020 en 68030 processoren. Een voor de hand liggende applicatie voor deze processoren in een UNIX-omgeving is een parallelle versie van *make*.[10] Het idee is hier dat als *make* ontdekt dat er meer compilaties nodig zijn, deze parallel op verschillende processoren worden uitgevoerd. Hoewel dit idee simpel klinkt, zijn er vele potentiële problemen. Om er één te noemen: om één enkele eindfile te maken moet een reeks van opeenvolgende commando's worden uitgevoerd en sommige hiervan zouden files kunnen gebruiken die door eerdere commando's worden gecreëerd. De oplossing die werd gekozen, is dat alle commando's parallel worden uitgevoerd, maar dat ze worden geblokkeerd als ze een file nodig hebben die wel aangemaakt wordt, maar nog niet volledig gegenereerd is.

Andere problemen hebben te maken met technische beperkingen van het *make*-programma. Bijvoorbeeld: omdat het verwacht dat commando's sequentieel worden uitgevoerd, in plaats van parallel, houdt het niet bij hoeveel processen het met *fork* heeft afgesplitst, terwijl dit aantal verscheidene systeemlimieten zou kunnen overschrijden. Tenslotte zijn er programma's, zoals *yacc*,[11] die hun uitvoer schrijven op files met een vaste naam, zoals *y.tab.c*. Als meerdere *yacc*'s draaien in dezelfde directory, dan schrijven ze naar dezelfde file, wat een rommeltje oplevert. Al deze problemen zijn op de een of andere manier behandeld, zoals beschreven door Baalbergen.[2]

De parallelle compilaties worden gedirigeerd door een nieuwe versie van *make*, *amake*. *Amake* gebruikt niet de traditionele *make-files*. De gebruiker vertelt welke source-files nodig zijn, maar niet de afhankelijkheden. De compilers zijn gewijzigd, zodat ze nu de geobserveerde afhankelijkheden bijhouden (bijvoorbeeld welke files ze in feite hebben geïncludeerd). Na een compilatie gaat deze informatie in een soort mini-gegevensbank die de traditionele *makefile* vervangt. Deze mini-gegevensbank houdt ook bij welke vlaggen worden gebruikt, welke versie van de compiler werd gebruikt en andere informatie. Dat niet eens meer gedacht hoeft te worden over *make-files*, zelfs niet over automatisch gegenereerde *make-files*, was populair bij de gebruikers. De overhead als gevolg van het management van de gegevensbank is verwaarloosbaar, maar de versnelling als gevolg van het gebruik van een parallelle methode is sterk afhankelijk van de invoer. Als men een programma maakt bestaande uit veel files van gemiddelde grootte, dan kan een aanzienlijke versnelling worden bereikt. Als een programma echter één grote source file heeft en veel kleintjes, dan kan de totale tijd nooit kleiner zijn dan de compilatietijd van de grote file.

5.3 Het handelsreizigersprobleem

Naast verschillende experimenten met de UNIX software hebben we ook geprobeerd sommige applicaties parallel te programmeren. Typerende applicaties zijn het handelsrei-

zigersprobleem[13] en de alfa/bèta zoekmethode.[14] We beschrijven deze hieronder kort. Meer details kunnen worden gevonden in Bal e.a.[3]

In het handelsreizigersprobleem worden aan de computer een startlokatie en een lijst met te bezoeken steden gegeven. Het idee is het kortste pad te vinden dat elke stad precies één keer langsgaat en terugkeert bij het beginpunt. Met Amoeba hebben we deze applicatie parallel geprogrammeerd door één pool-processor als coördinator te laten fungeren en de rest als slaven.
Stel, bijvoorbeeld, dat het startpunt Londen is en de verzameling te bezoeken steden bevat New York, Sydney, Nairobi en Tokio. De coördinator zou de eerste slaaf kunnen opdragen alle paden te onderzoeken die beginnen met Londen-New York; de tweede slaaf onderzoekt die, die beginnen met Londen-Sydney; de derde slaaf die, die beginnen met Londen-Nairobi, enzovoort. Al deze zoekpogingen vinden parallel plaats. Als een slaaf klaar is, rapporteert hij aan de coördinator en krijgt hij een nieuwe opdracht.
Het algoritme kan recursief worden toegepast. De eerste slaaf zou bijvoorbeeld een processor kunnen alloceren om alle paden te onderzoeken die beginnen met Londen-New York-Sydney, een andere processor voor Londen-New York-Nairobi, enzovoort. Natuurlijk is op een zeker moment een punt nodig waarop de slaaf de berekening echt zelf doet, in plaats van deze uit te besteden aan andere processoren.
De prestaties van het algoritme kunnen enorm worden verbeterd als het beste tot nu toe gevonden pad wordt onthouden. Een goed beginpad kan worden gevonden door de 'dichtstbijzijnde stad eerst'-heuristiek te gebruiken. Elke keer als een slaaf aan het werk wordt gezet, krijgt hij de lengte van het beste totale pad dat tot nu toe werd gevonden. Als hij ooit merkt dat hij werkt aan een partieel pad dat langer is dan het op dat moment beste totale pad, dan stopt hij onmiddellijk met waar hij mee bezig is, rapporteert dat zijn zoekpoging is gestrand en vraagt om meer werk. Eerste experimenten hebben getoond dat met dit algoritme 75 procent van de theoretisch maximale versnelling kan worden bereikt. De rest gaat verloren aan de kosten van communicatie.

5.4 De alfa/bèta-zoekmethode

Een andere applicatie die we parallel hebben geprogrammeerd met behulp van Amoeba, is het spelen van spelletjes met behulp van de alfa/bèta-heuristiek voor het snoeien van een zoekboom. Het algemene idee is hetzelfde als bij het handelsreizigersprobleem. Als een processor een bord te evalueren krijgt, genereert hij alle legale zetten die mogelijk zijn op dat bord en geeft deze aan andere processoren om ze parallel te evalueren.
De alfa/bèta-heuristiek wordt algemeen gebruikt in tweepersoons nul-somspelen om de zoekboom te snoeien. (Een tweepersoons nul-somspel is een spel dat door twee personen gespeeld wordt en waarbij de winst van de ene speler altijd gelijk is aan het verlies van de andere. Voorbeelden zijn schaken, dammen en go.)
Een raamwerk van waarden wordt gevormd en posities die buiten dit raamwerk vallen, worden niet onderzocht omdat bekend is dat er betere zetten bestaan. In tegenstelling tot het handelsreizigersprobleem, waar een groot deel van de boom moet worden afgezocht, is bij alfa/bèta een veel grotere beperking van de zoekboom mogelijk indien de posities in een goed gekozen volgorde worden geëvalueerd.
Op een enkele machine zouden we bijvoorbeeld op een zeker punt drie legale zetten A, B, en C kunnen hebben. Als een resultaat van de evaluatie van A zouden we kunnen ontdekken dat het kijken naar zijn verwanten in de boom, B en C, zinloos zou zijn. In een parallelle implementatie zouden we alles tegelijk doen en uiteindelijk de rekenkracht gewijd aan het onderzoeken van B en C verspillen. Het resultaat is dat een groot deel van het parallelle zoeken verspilde moeite is en het nettoresultaat is niet zoveel beter dan een sequentieel algoritme op een enkele processor. Onze experimenten met het draaien van Othello (Reversi) op Amoeba hebben getoond dat we niet in staat waren meer dan 40% van de totaal beschikbare processorcapaciteit te benutten, vergeleken met 75 procent voor het handelsreizigersprobleem.

6 Prestaties

Amoeba werd ontworpen om snel te zijn. Metingen laten zien dat dit doel werd bereikt. In deze paragraaf zullen we de resultaten presenteren van een aantal tijdmetingsexperimenten die we hebben gedaan. Deze metingen werden uitgevoerd op Sun 3/60s (20 MHz 68020s), gebruikmakend van een 10 Mbps Ethernet. We onderzochten de prestaties van drie verschillende configuraties:

1. twee gebruikersprocessen draaiend op Amoeba;
2. twee gebruikersprocessen draaiend op Sun OS 4.0.3, maar gebruik makend van de elementaire operaties van Amoeba;
3. twee gebruikersprocessen draaiend op Sun OS 4.0.3 en gebruik makend van Sun RPC.

De laatste twee configuraties waren alleen voor vergelijkingsdoeleinden bestemd. We deden tests voor het lokale geval (beide processen op dezelfde machine) en voor het remote geval (elk proces op een aparte machine, met communicatie over het Ethernet). In alle gevallen werd gecommuniceerd van proces naar proces en elk proces draaide in user-mode buiten de kernel. De meetresultaten representeren de gemiddelde waarden van 100.000 proefnemingen en zijn zeer goed reproduceerbaar.
Op elke configuratie (puur Amoeba, Amoeba-primitieven op UNIX, Sun RPC op UNIX) probeerden we drie test-

	Wachttijd (msec)			Brandbreedte (Kbytes/sec)		
	geval 1 (4 bytes)	geval 2 (8 Kb)	geval 3 (30 Kb)	geval 1 (4 bytes)	geval 2 (8 Kb)	geval 3 (30 Kb)
puur Amoeba lokaal	0.5	2.0	5.6	7.4	4000	5232
puur Amoeba remote	1.1	11.1	37.4	3.5	721	783
UNIX-driver lokaal	4.2	7.7	17.0	0.9	1039	1723
Unix-driver remote	5.1	26.7	88.6	0.8	300	331
Sun RPC lokaal	5.7	12.8	onmog.	0.7	625	onmog.
Sun RPC remote	6.7	24.6	onmog.	0.6	325	onmog.
		(*a*)			(*b*)	

Figuur 5: RPC tussen gebruikersprocessen in drie gebruikelijke gevallen voor drie verschillende systemen. Lokale RPC's zijn RPC's waarbij de cliënt en de server op dezelfde processor draaien; *(a)* wachttijd in msec; *(b)* bandbreedte in Kbytes/sec. De UNIX-driver implementeert Amoeba RPC's en Amoeba protocol onder Sun UNIX

gevallen te draaien: een boodschap van 4 bytes (1 integer), een boodschap van 8 Kbyte en een boodschap van 30 Kbyte. De boodschap van 4 bytes is typerend voor korte controleboodschappen, die van 8 Kbyte is typerend voor het lezen van een file van gemiddelde grootte van een remote file-server en de boodschap van 30 Kbyte is het maximum dat de huidige implementatie van Amoeba aankan. In totaal zouden we dus 9 gevallen moeten hebben (3 configuraties en 3 formaten). De standaard Sun RPC is echter beperkt tot 8K, dus we hebben metingen van maar 8 van de 9 gevallen. Ook moet opgemerkt worden dat de standaard Amoeba-header ruimte heeft voor 8 bytes aan gegevens, dus werd in de test met 4 bytes alleen een header gestuurd en geen data buffer. Aan de andere kant is er op de Sun een speciale optimalisatie beschikbaar voor het lokale geval, waarvan we gebruik hebben gemaakt.

In figuur 5 geven we de wachttijd en de bandbreedte van deze acht gevallen, zowel voor lokale processen (twee verschillende processen op dezelfde machine) als voor remote processen (processen op verschillende machines). De wachttijd is de tijd zoals die wordt gezien door de cliënt, draaiend als een gebruikersproces, tussen het aanroepen en terugkeren van de RPC-operatie. De bandbreedte is het aantal data-bytes per seconde dat de cliënt van de server ontvangt, de headers niet meegerekend. De metingen werden zowel voor lokale RPC's, waar cliënt- en server-processen draaiden op dezelfde processen, als voor remote RPC's, over Ethernet, verricht.

De interessante vergelijkingen in deze tabellen zijn de vergelijkingen van puur Amoeba RPC en puur Sun OS RPC voor zowel korte communicatie, waar wachttijd kritisch is, als lange, waar bandbreedte het punt is. Een Amoeba RPC van 4 bytes heeft 1.1 msec nodig, tegen 6.7 msec voor Sun RPC. Evenzo is voor RPC's van 8 Kbyte de bandbreedte van Amoeba 721 Kbytes/sec, tegen maar 325 Kbytes/sec voor de Sun RPC. De conclusie is dat de wachttijd van Amoeba 6 maal beter is en zijn throughput (de verwerkte hoeveelheid data) twee maal zo goed.

systeem	*hardware*	*lege RPC in msec.*	*throughput in kbytes/s*	*geschatte CPU MIPS*	*implementatie-opmerkingen*
Amoeba	Sun 3/60	1.1	783	3.0	gemeten user-naar-user
Cedar	Dorado	1.1	250	4.0	custom micro-code
x-Kernel	Sun 3/75	1.7	860	2.0	gemeten kernel-naar-kernel
V	Sun 3/75	2.5	546	2.0	gemeten user-naar-user
Topaz	Firefly	2.7	587	5.0	bestaat uit 5 VAX CPU's
Sprite	Sun 3/75	2.8	720	2.0	gemeten kernel-naar-kernel
Mach	Sun 3/60	11.0	?	3.0	throughput niet gerapporteerd

Figuur 6: Wachttijd en throughput voor een aantal systemen, zoals gerapporteerd in de literatuur

Hoewel de Sun uiteraard niet het enige interessante systeem is, is dit systeem door het wijdverbreide gebruik ervan een goede maatstaf. We hebben in de literatuur gezocht naar prestatiegegevens van andere gedistribueerde systemen en tonen de maximale throughput en de wachttijd voor een lege RPC in figuur 6.

De RPC-waarden van de andere systemen uit figuur 6 werden gehaald uit de volgende publikaties: Cedar,[5] *x*-Kernel,[19] Sprite,[18] V,[6] Topaz[22] en Mach.[19]

De hier getoonde getallen kunnen niet met elkaar worden vergeleken zonder kennis van de systemen waarop ze werden verkregen, omdat de snelheid van de hardware waarop de tests werden gedaan varieert tot een factor van ongeveer 3. Op alle gedistribueerde systemen van dit type die draaien op snelle LAN's, zijn de protocollen grotendeels gebonden aan de CPU. Het draaien van het systeem op een snellere CPU (maar met hetzelfde netwerk) verbetert zeker de prestaties, hoewel deze verbetering niet lineair is met CPU MIPS, omdat op een zeker punt het netwerk verzadigd raakt (hoewel geen van de hier opgesomde systemen ook maar in de buurt van verzadiging kwam). Als voorbeeld: in een eerder artikel[32] rapporteerden we een lege RPC-tijd van 1.4 msec, maar deze waarde hoorde bij Sun 3/50s. De huidige waarde van 1.1 msec geldt voor de snellere Sun 3/60s.

Figuur 6 is niet gecorrigeerd voor machinesnelheid, maar we hebben in ieder geval een ruwe schatting gemaakt van de ruwe totale rekenkracht van elk systeem, gegeven in de vijfde kolom van de tabel in MIPS (Miljoenen Instructies Per Seconde). Hoewel we ons realiseren dat dit op z'n best een grove maatstaf is, zien we geen andere manier om het feit te compenseren dat een systeem dat op een 4 MIPS machine (Dorado) of op een multiprocessor met 5 CPU's (Firefly) draait, een significant voordeel heeft ten opzichte van langzamere werkstations. Overigens: de Sun 3/60 is inderdaad sneller dan de Sun 3/75; dit is geen zetfout.

Cedars RPC is ongeveer hetzelfde als Amoeba's RPC, hoewel Cedars RPC werd geïmplementeerd op hardware die 33 procent sneller is. De throughput ervan is maar 30% van die van Amoeba, maar dat is gedeeltelijk het gevolg van het feit dat het Cedar-systeem gebruik maakt van een vroege versie van Ethernet, die draait op 3 megabits/sec. Aan de andere kant lukt het Cedars RPC niet eens de volledige 3 megabits/sec te benutten.

De *x*-Kernel heeft een 10% betere throughput dan Amoeba, maar de gepubliceerde metingen zijn kernel-naar-kernel, terwijl bij Amoeba werd gemeten van gebruikersproces naar gebruikersproces. Als de extra overhead van de contextwisselingen van kernel naar user en het kopiëren van kernel-buffers naar user-buffers in beschouwing wordt genomen om deze metingen vergelijkbaar te maken met de Amoeba-getallen, dan worden de *x*-kernel-prestaties gereduceerd tot 2.3 msec voor de lege RPC, met een throughput van 748 Kbytes/sec als inkomende gegevens worden afgebeeld van kernel naar user en 575 Kbytes/sec als ze worden gekopieerd (L. Peterson, privé-communicatie).

Evenzo zijn de gepubliceerde getallen van Sprite kernel-naar-kernel. Sprite ondersteunt geen RPC op gebruikersniveau, maar wat er in de buurt komt, is de tijd om een lege boodschap te sturen van het ene gebruikersproces naar een andere en het antwoord te ontvangen. De hiervoor benodigde tijd is 4.3 msec. De user-naar-user bandbreedte is 170 Kbytes/sec.[34]

V gebruikt een slimme techniek om zijn prestaties te verbeteren voor korte RPC's: de gehele boodschap wordt door het gebruikersproces in de CPU-registers gestopt en wordt er door de kernel uitgehaald voor verzending. Omdat de 68020 acht 4-byte data-registers heeft, kunnen op deze manier 32 bytes worden overgezonden.

Topaz RPC werd geïmplementeerd op Fireflies; dat zijn op VAX gebaseerde multiprocessoren. De prestaties die vermeld staan in figuur 6, kunnen alleen worden bereikt door aan elk uiteinde verschillende CPU's te gebruiken. Als slechts één enkele CPU aan elk eind wordt gebruikt, dan groeit de lege RPC-tijd tot 4.8 msec en valt de throughput terug naar 313 Kbytes/sec.

De lege RPC-tijd van Mach werd gehaald uit een artikel dat werd gepubliceerd in mei 1990[19] en is van toepassing op Mach 2.5, waarin de netwerkcode zich in de kernel bevindt. De RPC-prestaties van Mach zijn een factor van meer dan 3 slechter dan die van elk van de andere systemen en tien keer langzamer dan Amoeba. Een recentere meting aan een verbeterde versie van Mach geeft een RPC-tijd van 9.6 msec en een throughput van 250 Kbytes/sec (R. Draves, privé-communicatie).

Net als Amoeba zelf werd de Bullet Server ontworpen met als hoofddoel een hoge snelheid. Hieronder presenteren we een aantal metingen van wat bereikt is. De metingen werden gedaan met een Sun 3/60 cliënt die praatte met een remote Sun 3/60 file-server met een SCSI disk. Figuur 7 geeft de prestaties van de Bullet Server bij tests gedaan met files van 1 Kbyte, 16 Kbytes en 1 Mbyte. In de eerste ko-

Filegrootte	Wachttijd (msec)		Bandbreedte (Kbyte/sec)	
	READ	CREATE+DEL	READ	CREATE+DEL
1 Kbyte	2	50	427	20
16 Kbyte	20	84	788	191
1 Mbyte	1260	3210	813	319
	(*a*)		(*b*)	

Figuur 7: Prestaties van de Bullet file-server bij read-operaties en create- en delete-operaties samen; *(a)* wachttijd in msec.; *(b)* bandbreedte in Kbytes/sec.

Filegrootte	Wachttijd (msec) READ	Wachttijd (msec) CREATE+DEL	Bandbreedte (Kbyte/sec) READ	Bandbreedte (Kbyte/sec) CREATE+DEL
1 Kbyte	20	101	50	10
16 Kbyte	47	186	340	86
1 Mbyte	2030	13350	504	77
	(a)		(b)	

Figuur 8: Prestaties van de Sun NFS file-server bij read- en create-operaties; *(a)* wachttijd in msec.; *(b)* Bandbreedte in Kbytes/sec.

lom worden de wachttijd en bandbreedte voor lees-operaties getoond. Merk op dat de testfile volledig in het geheugen zal zijn en dat geen disk access nodig is. In de tweede kolom wordt een combinatie van een create- en delete-operatie gemeten. In dit geval wordt de file naar disk geschreven. Merk op dat voor zowel de create- als de delete-operatie diskhandelingen nodig zijn.

De oplettende lezer kan hebben gemerkt dat een gebruikersproces 813 Kbytes/sec uit de server kan halen (uit figuur 7), hoewel de user-naar-user bandbreedte maar 783 Kbytes/sec is (uit figuur 5). De oorzaak van deze ogenschijnlijke discrepantie is de volgende. Wat de cliënten betreft is de Bullet Server gewoon een zwarte doos. Hij accepteert verzoeken en geeft antwoord. Op zijn machine draaien geen gebruikersprocessen. Vanwege deze omstandigheden besloten we de code van de Bullet Server in de kernel te stoppen, omdat de gebruikers toch het verschil niet kunnen zien en protectie een probleem is dat niet speelt op een losstaande file-server met maar één proces. De waarde van 813 Kbytes/sec is dus user-naar-kernel voor toegang tot de file cache, terwijl de waarde van 783 Kbytes/sec geldt voor user-naar-user communicatie, van geheugen naar geheugen, zonder dat er files in het spel zijn. De pure user-naar-kernel bandbreedte is zeker hoger dan 813 Kbytes/sec, maar een gedeelte ervan gaat verloren aan file-server overhead.

Om de resultaten van Amoeba te vergelijken met het Sun NFS filesysteem, hebben we het lezen en creëren van files gemeten op een Sun 3/60, gebruik makend van een remote Sun 3/60 file-server met 16 Mbyte geheugen, draaiend onder SunOS 4.0.3. De file-server had hetzelfde type disk als de Bullet Server, dus de hardware-configuraties waren, met uitzondering van het extra geheugen voor NFS, identiek aan die, die werden gebruikt om metingen te verrichten aan Amoeba. De metingen werden 's nachts uitgevoerd onder een lage belasting. Om lokale caching op de Sun 3/60 uit te schakelen, deden we de file op slot met behulp van de elementaire Sun UNIX-operatie *lockf* terwijl we de leestest deden. De tijdmeting van de leestest bestond uit herhaalde metingen aan een *seek* gevolgd door een *read* system-call. De schrijftest bestond uit het achtereenvolgens uitvoeren van *create*, *write* en *close*. (De *create* heeft het effect dat de vorige versie van de file verwijderd wordt.) De resultaten worden in beeld gebracht in figuur 8.

Merk op dat het lezen en creëren van 1 Mbyte files resulteert in lagere bandbreedten dan bij het lezen en creëren van 16 Kbyte files. De prestaties van de Bullet file-server bij leesoperaties zijn twee tot drie maal beter dan die van de Sun NFS file-server. Voor creëer-operaties heeft de Bullet file-server een constante overhead voor de produktie van capabilities, waardoor hij relatief beter presteert bij grote files.

7 Evaluatie

In deze paragraaf zullen we een kritische blik werpen op Amoeba en zijn evolutie. We zullen een aantal aspecten noemen die we succesvol vinden en andere die we minder succesvol vinden. Op gebieden waar Amoeba 4.0 tekort bleek te komen zullen we verbeteringen aanbrengen in Amoeba 5.0, dat op het moment in ontwikkeling is. Deze verbeteringen worden hieronder besproken.

Een gebied waarop weinig verbetering nodig is, is *portability* (overdraagbaarheid). Amoeba begon op de 680 × 0 CPU's en is gemakkelijk verplaatst naar de VAX, NS 32016 en Intel 80386. Het Amoeba RPC protocol is ook geïmplementeerd als een deel van MINIX[25] en wordt als zodanig op grote schaal gebruikt over de hele wereld.

7.1 Objecten en capabilities

Over het algemeen werkte het basisidee van een op objecten gebaseerd systeem goed. Het gaf ons een kader waarbinnen het gemakkelijk is over het systeem na te denken. Als nieuwe objecten of services worden voorgesteld, hebben we te maken met een duidelijk model en moeten we specifieke vragen beantwoorden. We moeten in het bijzonder voor elke nieuwe service beslissen welke objecten zullen worden ondersteund en welke operaties op deze objecten zullen worden toegestaan. Deze structureringstechniek is bij veel gelegenheden waardevol gebleken.

Het gebruik van *capabilities* om een naam te geven aan objecten en ze te beschermen is ook een succes gebleken. Door het gebruik van cryptografisch beschermde capabilities hebben we een unieke naam van vaste lengte voor elk object over het hele systeem, wat resulteert in een hoge mate van transparantie. Op die manier is het simpel een basale directory te implementeren als een verzameling (ASCII-string, capability)-paren. Als resultaat hiervan kan een directory namen bevatten voor veel verschillende soorten objecten, die zich overal op de wereld kunnen be-

vinden, en kan op windows worden geschreven door elk proces dat de bijbehorende capability bezit, onafhankelijk van de lokatie waarop dat proces zich bevindt. Wij vinden dit model zowel simpeler als flexibeler dan modellen die gebruik maken van remote mounting en symbolic links, zoals Suns NFS. Bovendien kan het net zo efficiënt worden geïmplementeerd.

We hebben geen ervaring met capabilities op grote systemen (met duizenden gebruikers tegelijkertijd). Aan de ene kant zit het er met zo'n groot systeem in dat sommige capabilities uitlekken, waardoor de veiligheid in gevaar komt. Aan de andere kant vormen capabilities een soort brandmuur, aangezien een gecompromitteerde capability alleen de veiligheid van één object aantast. Of zo'n fijnkorrelige bescherming in de praktijk beter of slechter is dan conventionele beveiligingsmethoden voor grote systemen, is op dit moment moeilijk te zeggen.

We zijn ook tevreden met de *gebruikersprimitieven* op laag niveau. Feitelijk zijn er maar drie belangrijke system-calls, *get_request*, *put_reply* en *do_operation*, die elk gemakkelijk te begrijpen zijn. Alle communicatie wordt op deze elementaire operaties gebaseerd, die veel eenvoudiger zijn dan bijvoorbeeld de 'socket interface' in Berkeley UNIX, met zijn talloze system-calls, parameters en opties.
Amoeba 5.0 zal gebruik maken van 256-bit capabilities, in plaats van de 128-bit capabilities van Amoeba 4.0. Het grotere controleveld zal beter bestand zijn tegen aanvallen. Andere veiligheidsaspecten zullen ook strakker worden aangepakt, waaronder de toevoeging van veilige, gecodeerde communicatie tussen cliënt en server. Tevens zullen de grotere capabilities ruimte hebben voor een *lokatie-hint*, die door de SWAN servers kan worden gebruikt voor het lokaliseren van objecten in het wide-area netwerk. Ten derde zullen alle velden van de nieuwe 256-bit capabilities uitgelijnd worden op 32-bit grenzen, wat in potentie betere prestaties zou kunnen opleveren.

7.2 Remote procedure call
Voor het grootste deel verloopt RPC-communicatie bevredigend, maar soms levert ze problemen op.[27] In het bijzonder is RPC inherent meester-slaaf en punt-naar-punt. Soms leiden beide van deze eigenschappen tot problemen. In een UNIX pipeline, zoals

pic file | eqn | tbl | troff>outfile

bijvoorbeeld, is er geen inherente meester/slaaf-relatie en is het helemaal niet duidelijk of het verplaatsen van data tussen de elementen van de pipeline gecontroleerd moet worden door de lezer (read driven) of door de schrijver (write driven) van de data.

Als een RPC een lange boodschap overzendt in Amoeba 4.0, dan wordt in feite een serie pakketten gestuurd, die elk individueel worden bevestigd op het niveau van de driver (stop-and-wait-protocol). Hoewel dit ontwerp simpel is, vertraagt het het systeem. In Amoeba 5.0 zullen we alleen gehele boodschappen bevestigen, wat ons in staat zal stellen hogere bandbreedtes te bereiken dan die, die werden getoond in figuur 6.

Omdat RPC inherent punt-naar-punt is, treden problemen op in parallelle applicaties als het handelsreizigersprobleem. Als een proces een pad ontdekt dat beter is dan het op dit moment bekende beste pad, dan wil het eigenlijk een *multicast*-boodschap sturen aan een groot aantal processen om ze er allemaal onmiddellijk van op de hoogte te stellen. Op het moment is dit onmogelijk en moet het òf worden gesimuleerd met meerdere RPC's, òf met een foefje worden opgelost.
Amoeba 5.0 zal groepscommunicatie en multicasting volledig ondersteunen. Een boodschap die aan een groep wordt gestuurd, zal bij alle leden van de groep worden bezorgd, of althans een poging daartoe zal worden ondernomen. Er is een protocol voor een hoger niveau bedacht om 100% betrouwbare multicasting te implementeren op onbetrouwbare netwerken, tegen nagenoeg dezelfde prijs als die van RPC (twee boodschappen per betrouwbare broadcast). Dit protocol wordt beschreven door Kaashoek e.a.[12] Voor veel applicaties (bijvoorbeeld verschillende soorten gedupliceerde gegevensbanken) maakt betrouwbare broadcasting het leven een stuk eenvoudiger. Amoeba 5.0 zal deze duplicatiefaciliteit gebruiken om fouttolerantie te ondersteunen.

Hoewel niet elk LAN broadcasting en multicasting met hardware ondersteunt, kan het, als het beschikbaar is (zoals in Ethernet), een enorme prestatiewinst opleveren voor veel applicaties. Bijvoorbeeld: een eenvoudige manier om een wijziging aan te brengen in een gedupliceerde gegevensbank is een betrouwbare multicast te sturen naar alle machines waarop zich een kopie van de gegevensbank bevindt. Dit idee is voor de hand liggend. We hadden het ons eerder moeten realiseren en het vanaf het begin in het systeem moeten stoppen.

Het is gelukkig allang weer gecorrigeerd, maar in Amoeba 2.0 hadden we de werkelijk verschrikkelijke beslissing genomen RPC *asynchroon* te maken. In dat systeem stuurde de zender een boodschap aan de ontvanger en ging dan verder met het uitvoeren van zijn programma. Als het antwoord binnen kwam, werd de zender geïnterrumpeerd. Deze opzet liet een aanzienlijke mate van parallelle verwerking toe, maar het was onmogelijk om het correct te programmeren. Ons advies aan toekomstige ontwerpers is, asynchrone boodschappen te mijden als de pest.

7.3 Management van geheugen en processen

Waarschijnlijk was de grootste vergissing in het ontwerp van de procesmanagementmechanismen van Amoeba 4.0, de beslissing de threads hun taak *altijd* te laten voltooien (dat wil zeggen ze zijn 'niet pre-emptable'). Het idee was dat een thread, als hij eenmaal begonnen was een kritische tabel te gebruiken, niet zou worden onderbroken door een andere thread in hetzelfde proces, totdat hij logisch blokkeerde. Deze opzet leek eenvoudig te begrijpen en was zeker eenvoudig te programmeren.

Problemen ontstonden omdat programmeurs niet een erg goed beeld hadden van de vraag *wanneer* een proces blokkeerde. Bijvoorbeeld: om code in een kritisch gebied te debuggen zou een programmeur wat print-opdrachten kunnen toevoegen temidden van de code in het kritisch gebied. Deze print-opdrachten zouden bibliotheekroutines kunnen aanroepen die RPC's deden met een remote terminal server. Terwijl hij geblokkeerd was en wachtte op een bevestiging, zou een thread kunnen worden onderbroken en een andere thread zou toegang kunnen krijgen tot het kritisch gebied en verwoestingen aanrichten. Op die manier zou de veiligheid van het kritische gebied kunnen worden vernietigd door het toevoegen van print-opdrachten. Deze eigenschap was erg verwarrend voor naïeve programmeurs.

De 'run-to-completion'-semantiek van de thread-coördinatie in Amoeba 4.0 maakt het een multiprocessor-implementatie ook onmogelijk parallelle verwerking en gedeeld geheugen uit te buiten door verschillende threads in één proces aan verschillende processoren toe te wijzen. De threads van Amoeba 5.0 zullen parallel kunnen draaien. Door de coördinator worden geen beloftes gedaan dat een thread mag draaien tot hij blokkeert voordat een andere thread wordt uitgekozen om te draaien. Threads die resources delen, moeten expliciet met elkaar synchroniseren met behulp van semaforen of mutexes.

Een volgend probleem is dat de duur van remote operations niet aan time-outs gebonden is. Als de geheugenserver een proces begint, gebruikt hij de capabilities in de proces-descriptor om de code en data in te laden. Het is volkomen legaal als deze capabilities horen bij iemands privé file-server, in plaats van bij de Bullet Server. Als deze server echter kwaad wil en gewoon helemaal niet antwoordt, dan zal een thread in de geheugen-server gewoon voor altijd hangen. We hadden waarschijnlijk service time-outs moeten toevoegen, hoewel dat race-condities zou hebben geïntroduceerd.

Tenslotte ondersteunt Amoeba geen *virtueel geheugen*. Het is en was onze werkaanname dat geheugen zo goedkoop aan het worden is, dat het de extra complexiteit die virtueel geheugen met zich meebrengt, niet waard is. Tegenwoordig hebben de meeste werkstations 4M RAM en binnen een aantal jaren zullen ze 32M hebben. De eenvoud van ontwerp en implementatie en het bereiken van hoge snelheden zijn altijd onze doelen geweest, dus we hebben nog niet besloten of virtueel geheugen geïmplementeerd zal worden in Amoeba 5.0.

In dezelfde geest ondersteunen we procesmigratie op het ogenblik niet, hoewel de daarvoor benodigde mechanismen al bestaan. Of procesmigratie voor het uitbalanceren van de belasting een essentieel onderdeel of gewoon een extra fraaiigheid is, staat nog ter discussie.

7.4 Filesysteem

Eén gebied van het systeem waarvan we vinden dat het uitzonderlijk succesvol is gebleken, is het ontwerp van de fileserver en de directory-server. We hebben twee aparte delen onderscheiden, de Bullet Server, die alleen opslag verzorgt, en de directory-server, die naamgeving en bescherming verzorgt. Het ontwerp van de Bullet Server maakt hem extreem snel, terwijl het ontwerp van de directory-server een flexibel protectiesysteem levert en tevens fileduplicatie op een simpele en eenvoudig te begrijpen manier ondersteunt. Het sleutelelement is hier dat files onveranderbaar zijn, zodat ze kunnen worden gedupliceerd wanneer gewenst en kopieën opnieuw kunnen worden gegenereerd wanneer nodig.

Het volledige duplicatieproces vindt plaats op de achtergrond ('luie replicatie') en gebeurt volledig automatisch, zodat de gebruiker er helemaal niet mee belast wordt. We achten het filesysteem het meest vernieuwende onderdeel van het ontwerp van Amoeba 4.0, omdat het goede prestaties combineert met betrouwbaarheid, robuustheid en gebruiksgemak.

Een onderwerp waarin we nu geïnteresseerd raken, is hoe men zou kunnen omgaan met gegevensbanken in deze omgeving. We stellen ons een op Amoeba gebaseerd gegevensbanksysteem voor dat een zeer groot geheugen zou hebben voor een gegevensbank die in essentie 'in-core' is. Updates zouden in het geheugen worden gedaan. De enige functie van de disk zou zijn periodiek checkpoints te maken. Op deze manier zou de onveranderbaarheid van de files geen problemen opleveren.

Een probleem dat nog niet is opgetreden, maar zou kunnen optreden als Amoeba zou worden uitgebreid tot duizenden gebruikers, wordt veroorzaakt door het opsplitsen van de directory-server en de file-server. Het creëren van een file en het vervolgens toevoegen van z'n capability aan een directory zijn twee afzonderlijke operaties. Als de cliënt tussen deze operaties in zou uitvallen, bestaat de file, maar is hij niet toegankelijk. Onze huidige strategie is dat de directory-server elke file die hij kent ongeveer eens in elke *k* dagen langsgaat en dat de Bullet Server automatisch door middel van garbage-collection alle files verwijdert die

door niemand in de afgelopen *n* dagen werden gebruikt ($n >> k$). In onze huidige opstelling en met betrouwbare hardware is dit geen probleem, maar in een groot, internationaal Amoeba-systeem zou het dat kunnen worden.

7.5 Onderlinge verbinding van netwerken

We zijn ook blij met de manier waarop het gebruik van wide-area netwerken is geregeld, met behulp van server-agenten, cliënt-agenten en de SWAN. In het bijzonder is het feit dat het bestaan van wide-area netwerken de protocollen en prestaties van lokale RPC's niet beïnvloedt, van cruciaal belang. Veel andere ontwerpen (bijvoorbeeld TCP/IP, OSI) beginnen met het wide-area-geval en gebruiken dit vervolgens ook op het lokale vlak. Deze keuze resulteert in significant slechtere prestaties op een LAN dan het Amoeba-ontwerp en in niet-betere prestaties over wide-area netwerken.

Eén configuratie waar Amoeba 4.0 niet adequaat mee omgaat, is een systeem bestaande uit een groot aantal local-area netwerken, onderling verbonden door vele bridges en gateways. Hoewel Amoeba 4.0 werkt op deze systemen, zijn z'n prestaties slecht, deels vanwege de manier waarop de lokalisering van ports en het behandelen van boodschappen wordt gedaan. In Amoeba 5.0 hebben we een volledig nieuw laag-niveau protocol ontworpen en geïmplementeerd, genaamd *Fast Local Internet Protocol* (FLIP), dat de prestaties op complexe onderling verbonden netwerken enorm zal verbeteren. FLIP heeft meer facetten, maar onder andere zullen volledige boodschappen worden bevestigd, in plaats van individuele pakketten, wat het aantal te verwerken interrupts enorm vermindert. Lokalisering van ports wordt ook efficiënter gedaan en een enkele server-agent kan nu luisteren naar een arbitrair aantal ports, wat het aantal server-agenten in ruste dat in de gateways voor grote systemen nodig is, enorm zal verminderen.

Eén onverwacht probleem dat we hadden, was de slechte *kwaliteit van de wide-area netwerken* die we moesten gebruiken, in het bijzonder de publieke X.25-netwerken. Ook moesten we vaak diverse netwerken overbruggen om toegang te krijgen tot sommige machines, ieder met zijn eigen problemen en eigenaardigheden. Ons enige inzicht voor toekomstige onderzoekers is, niet blindelings aan te nemen dat wide-area netwerken werkelijk correct functioneren totdat dat experimenteel is geverifieerd.

7.6 UNIX-emulatie

De UNIX-emulatie van Amoeba 4.0 bestaat uit een bibliotheek en een session-server. De emulatie werd geschreven met het idee zoveel mogelijk van de UNIX software aan het werk te krijgen zonder al te veel moeite van onze kant. De prijs die we voor deze aanpak betalen, is dat we nooit 100% compatibiliteit kunnen leveren. Het hele concept van user-ids en group-ids is bijvoorbeeld erg moeilijk goed in een op capabilities gebaseerd systeem te verwerken. Onze kijk op bescherming is volledig anders. Daarnaast is Amoeba in essentie een toestandsloos systeem. Dit betekent dat verschillende subtiele eigenschappen van UNIX met betrekking tot de wijze waarop files worden gedeeld door ouder en kind, haast onmogelijk in orde te maken zijn. In de praktijk kunnen we hiermee leven, maar voor iemand die compatibiliteit op binair niveau eist, heeft onze aanpak een aantal tekortkomingen.

7.7 Parallelle applicaties

Hoewel Amoeba oorspronkelijk werd gezien als een systeem voor *gedistribueerd* rekenen, maakt het bestaan van een processor-pool met 48 × 68020 CPU's dicht bij elkaar het systeem ook redelijk geschikt voor *parallel* rekenen. Dat wil zeggen dat we veel meer geïnteresseerd zijn geraakt in het gebruik van de processor-pool voor het bereiken van grote versnellingen voor een enkel probleem. Om deze parallelle applicaties te programmeren zijn we op het ogenblik bezig een taal te implementeren genaamd *Orca*.[4]

Orca is gebaseerd op het concept van globaal gedeelde objecten. Programmeurs kunnen operaties op gedeelde objecten definiëren en de compiler en het run-time-systeem zorgen voor alle details en verzekeren dat ze correct worden uitgevoerd. Deze opzet geeft de programmeur de mogelijkheid atomair gedeelde objecten te lezen en te schrijven die fysiek zijn gedistribueerd over een verzameling machines, zonder dat hij zich bezig hoeft te houden met de complexiteit van de fysieke distributie. Alle details van de fysieke distributie worden volledig voor de programmeur verborgen. De eerste resultaten geven aan dat een bijna lineaire versnelling kan worden bereikt bij problemen waarbij 'branch and bound', 'successive overrelaxation' en graafalgoritmen een rol spelen. We hebben bijvoorbeeld het handelsreizigersprobleem in Orca overgedaan en een tienvoudige versnelling bereikt met 10 processoren (vergeleken met 7.5, gebruik makend van de non-Orca-versie die eerder werd beschreven). De alfa/bèta-zoekmethode in Orca krijgt een versnelling met een factor 6 met 10 processoren (vergeleken met 4 zonder Orca). Het lijkt dat het gebruik van Orca de communicatie-overhead reduceert, maar het blijft waar dat er altijd moeilijkheden zullen optreden bij problemen met veel processen en een hoge graad van interactie (dus een kleine korrelgrootte).

7.8 Prestaties

In het algemeen zijn de prestaties een groot succesverhaal geweest. De minimum RPC-tijd voor Amoeba is 1.1 msec tussen twee processen in user-space op Sun 3/60s en de interproces-throughput is bijna 800 Kbytes/sec. Het filesysteem laat ons files lezen en schrijven met ongeveer dezelfde snelheid.

7.9 Gebruikersinterface

Amoeba had oorspronkelijk een eigengemaakt window-systeem. Het was sneller dan X-windows en in onze optiek netter. Het was ook veel kleiner en eenvoudiger te begrijpen. Vanwege deze redenen dachten we dat acceptatie door de gebruikers geen probleem zou zijn. Dat was niet zo. Technische factoren spelen soms tweede viool naast factoren van politiek en marketing. We hebben onze window-server losgelaten en zijn overgestapt naar X-windows.

7.10 Veiligheid

Een indringer die in staat is het netwerk waar Amoeba op draait, af te tappen, kan capabilities ontdekken en aanzienlijke schade aanrichten. In een produktie-omgeving is een vorm van encryptie van data op het netwerk nodig om een grotere veiligheid te garanderen. Hoewel enkele gedachten zijn gewijd aan een veiligheidsmechanisme,[26] werd het niet in Amoeba 4.0 geïmplementeerd.

Twee potentiële veiligheidssystemen zijn ontworpen voor Amoeba 5.0. De eerste versie kan alleen worden gebruikt in vriendelijke omgevingen waar van het netwerk en de bedrijfssysteem-kernels kan worden aangenomen dat ze veilig zijn. Deze versie gebruikt éénrichtingscodes en zou, met caching van argument/resultaat-paren, vrijwel net zo snel kunnen draaien als de huidige Amoeba-versie. De andere versie maakt geen aannamen omtrent de veiligheid van het onderliggende netwerk of het bedrijfssysteem. Net als MIT's Kerberos[23] gebruikt deze versie een door iedereen vertrouwde verificatie-server voor het vervaardigen en verspreiden van sleutels en wordt al het netwerkverkeer gecodeerd.

We zijn van plan beide versies te installeren en de effecten op de prestaties van het systeem te onderzoeken. We onderzoeken de problemen van verificatie in zeer grote systemen die verschillende organisaties omvatten en landsgrenzen overschrijden.

8 Vergelijking met andere systemen

Amoeba is niet het enige gedistribueerde systeem in de wereld. Bekende systemen zijn onder andere Mach,[1] Chorus,[21] V,[6] en Sprite.[18] Hoewel een uitgebreide vergelijking van Amoeba met deze systemen zonder twijfel interessant zou zijn, valt dit buiten het bereik van dit artikel. Desalniettemin willen we graag een paar algemene opmerkingen maken.

Het hoofddoel van het Amoeba-project verschilt enigszins van de doelen van de meeste andere systemen. Het was onze bedoeling een nieuw bedrijfssysteem van de grond af op te bouwen, gebruik makend van de beste ideeën die op het moment beschikbaar zijn, zonder aandacht te besteden aan compatibiliteit met systemen die 20 jaar geleden werden ontworpen. Meer in het bijzonder was 100% compatibiliteit met UNIX nooit een doel, hoewel we een bibliotheek en een server hebben geschreven die genoeg UNIX-compatibiliteit verschaffen om meer dan 100 UNIX-utilities op Amoeba te laten draaien (na opnieuw linken met een speciale bibliotheek). Hoewel het niet mikken op volledige compatibiliteit met de laatste UNIX-versie vanuit een marketing-gezichtspunt potentiële klanten met grote bestaande softwarebanken zou kunnen afschrikken, is de vrijheid om op een selectieve manier om te gaan met de ideeën uit UNIX uit onderzoeks-oogpunt een voordeel. Sommige andere systemen nemen een ander gezichtspunt in.

Nog een verschil met andere systemen is onze nadruk op Amoeba als een *gedistribueerd* systeem. Van het begin af aan was de intentie het te laten draaien op een groot aantal machines. Eén vergelijking met Mach is op dit punt leerzaam. Mach gebruikt een slimme optimalisatie om boodschappen door te geven tussen processen die op dezelfde machine draaien. De pagina die de boodschap bevat, wordt afgebeeld van de adresseringsruimte van de zender naar die van de ontvanger, waardoor kopiëren vermeden wordt. Amoeba doet dit niet, omdat wij de communicatiesnelheid tussen processen die draaien op verschillende machines het hoofdpunt vinden. Dat is het normale geval. Slechts zelden zullen twee processen zich in een werkelijk gedistribueerd systeem op dezelfde fysieke processor blijken te bevinden, zeker als er honderden processoren zijn, en dus hebben we veel moeite gestopt in optimalisatie van het gedistribueerde geval, niet in het lokale geval. Dit is duidelijk een verschil in filosofie.

9 Conclusie

Het Amoeba-project heeft duidelijk aangetoond dat het mogelijk is een efficiënt gedistribueerd bedrijfssysteem met hoge prestaties te bouwen op huidige hardware. De op objecten gebaseerde aard van het systeem en het gebruik van capabilities zorgen voor een verenigend thema dat de verschillende onderdelen aan elkaar bindt. Door de kernel zo klein mogelijk te maken, worden de meeste sleuteleigenschappen geïmplementeerd als gebruikersproces, wat betekent dat het systeem geleidelijk kan evolueren, al naar gelang de behoeften zich wijzigen en we meer leren over gedistribueerd rekenen.

Amoeba werkt al enige jaren bevredigend, zowel lokaal als tot op zekere hoogte over een wide-area netwerk. Het ontwerp is netjes en de prestaties zijn zeer goed. We zijn meer dan tevreden met de resultaten. Desalniettemin is geen enkel bedrijfssysteem ooit klaar en dus werken we er continu aan om het te verbeteren. Amoeba is nu beschikbaar. Voor informatie over de wijze waarop het verkrijgbaar is, kan contact worden opgenomen met de eerste auteur (Tanenbaum), liefst via electronic mail.

Als geheugensteuntje: een beknopte lijst van in dit artikel gebruikte Engelse termen

branch and bound: bepaalde zoekstrategie.
bridge: brug (hardware-verbinding tussen twee netwerken).
cache: voor de programmeur onzichtbaar snel tussengeheugen.
checkpoint: controlepunt.
concurrent access: gelijktijdige toegang.
core image: geheugenbeeld.
directory: indexbestand.
DMA: directe geheugeninvoer.
garbage-collection: het verwijderen van lege posities uit het geheugen.
gateway: knooppunt dat de verbinding vormt tussen twee netwerken.
in-core: in het werkgeheugen.
kernel: deel van het besturingssysteem.
mailbox: postbus.
make: systeem om programma's die uit meer modulen bestaan te beheren.
multicasting: het versturen van berichten naar een aantal computers.
multiway-gateway: verbinding tussen een aantal netwerken.
mutex: een primitief om parallelle processen te synchroniseren.
port: toegang (software-verbinding tussen processen).
port-locate: het opzoeken van een port.
pre-emptable: met de mogelijkheid tot tijdelijk onderbreken (re-schedulen) van een proces voordat dit klaar is.
relayeren: doorsturen.
remote: op afstand.
remote mount: het toegankelijk maken van een filesysteem van een andere machine.
run-to-completion: een proces door laten draaien tot het klaar is.
SCSI: Small Computer System Interface.
service time-out: het aflopen van de blokkeertijd van een service.
singleton: verzameling bestaande uit één element.
socket interface: verbindingspunt tussen twee of meer processen.
stack: stapel.
stackpointer: stapelverwijzer.
stop-and-wait: netwerkprotocol waarbij de zender altijd wacht op bevestiging van de ontvanger.
stub: vervanger.
successive overrelaxation: techniek voor het oplossen van Laplace-vergelijkingen.
symbolic link: symbolische verwijzing naar een bestand.
system-call: aanroep van procedure binnen besturingssysteem.
system-call interface: manier waarop procedures van een besturingssysteem worden aangeroepen.
thread: lichtgewicht proces.
time-out: blokkeertijd.
timesharing: afwisselend gebruik van verwerkingstijd van een computer.
user-mode: toestand waarin het systeem verkeert als een gebruikersproces draait.
wide-area: wijd (netwerk).

Literatuur

1 Accetta, M., R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian en M. Young (1986), 'Mach: a new kernel foundation for UNIX development'. *Proceedings of the Summer Usenix Conference*, Atlanta (GA), July 1986.
2 Baalbergen, E.H., K. Verstoep en A.S. Tanenbaum (1989), 'On the design of the Amoeba configuration manager', *Proceedings 2nd Int'l Workshop on Software Configuration Management*, ACM.
3 Bal, H.E., R. van Renesse en A.S. Tanenbaum (1987), 'Implementing distributed algorithms using remote procedure call', *Proceedings National Computer Conference*, AFIPS, blz. 499-505.
4 Bal, H.E. en A.S. Tanenbaum (1988), 'Distributed programming with shared data', *IEEE Conference on Computer Languages*, IEEE, blz. 82-91.
5 Birrell, A.D. en B.J. Nelson (1984), 'Implementing remote procedure calls', *ACM Transactions on Computer Systems* 2 (februari 1984), blz. 39-59.
6 Cheriton, D.R. (1988), 'The V Distributed System', *Communications of the ACM* 31 (maart 1988), blz. 314-333.
7 Dalal, Y.K. (1977), *Broadcast protocols in packet switched computer networks*, Ph.D. Thesis, Stanford University.
8 Dennis, J. en E. van Horn (1966), 'Programming semantics for multiprogrammed computation', *Communications of the ACM* 9 (maart 1966), blz. 143-155.
9 Evans, A., W. Kantrowitz en E. Weiss (1974), 'A user authentication scheme not requiring secrecy in the computer', *Communications of the ACM* 17 (augustus 1974), blz. 437-442.
10 Feldman, S.I. (1979), 'Make – A program for maintaining computer programs', *Software – Practice and Experience* 9 (april 1979), blz. 255-265.
11 Johnson, S.C. (1978), *Yacc, Yet Another Compiler Compiler*, Bell Labs Technical Report, Bell Labs, Murray Hill (New Jersey).
12 Kaashoek, M.F., A.S. Tanenbaum, S. Flynn Hummel en H.E. Bal (1989), 'An efficient reliable broadcast protocol', *Operating Systems Review*, Vol. 23 (oktober 1989), blz. 5-19.
13 Lawler, E.L. en D.E. Wood (1966), 'Branch and bound methods, a survey', *Operations Research* 14 (juli 1966), blz. 699-719.
14 Marsland, T.A. en M. Campbell (1982), 'Parallel search of strongly ordered game trees', *Computing Surveys* 14 (december 1982), blz. 533-551.
15 Mullender, S.J., G. van Rossum, A.S. Tanenbaum, R. van Renesse en J.M. van Staveren, 'Amoeba – A distributed operating system for the 1990s', *IEEE Computer* 23 (mei 1990), blz. 44-53.
16 Mullender, S.J. en A.S. Tanenbaum (1986), 'The design of a capability-based distributed operating system, *Computer Journal* 29 (augustus 1986), blz. 289-299.

17 Mullender, S.J. en A.S. Tanenbaum (1985), 'A distributed file service based on optimistic concurrency control, *Proceedings of the Tenth Symposium on Operating System Principles* (december 1985), blz. 51-62.
18 Ousterhout, J.K., A.R. Cherenson, F. Douglis, M.N. Nelson en B.B. Welch (1988), 'The Sprite network operating system', *IEEE Computer* 21 (februari 1988), blz. 23-26.
19 Peterson, L., N. Hutchinson, S. O'Malley en H. Rao, 'The *x*-kernel: A platform for accessing internet resources', *IEEE Computer* 23 (mei 1990), blz. 23-33.
20 Pu, C., J.D. Noe, A. Proudfoot (1986), 'Regeneration of replicated objects: A technique and its Eden implementation', *Proceedings 2nd International Conference on Data Engineering* (februari 1986), blz. 175-187.
21 Rozier, M., V. Abrossimov, F. Armand, I. Boule, M. Gien, M. Guillemont, F. Hermann, C. Kaiser, S. Langlois, P. Leonard en W. Neuhauser, 'CHORUS distributed operating system', *Computing Systems* 1 (Fall 1988), blz. 299-328.
22 Schroeder, M.D. en M. Burrows, 'Performance of the Firefly RPC', *Proceedings of the Twelfth ACM Symposium of Operating Systems Principles*, ACM (december 1989), blz. 83-90.
23 Steiner, J.G., C. Neuman, C. en J.I. Schiller (1988), 'Kerberos, an authentication service for open network systems', *Proceedings of the Usenix Winter Conference*, USENIX Association, blz. 191-201.
24 Stonebraker, M. (1981), 'Operating system support for database management', *Communications of the ACM* 24 (juli 1981), blz. 412-418.
25 Tanenbaum, A.S. (1987), 'A UNIX clone with source code for operating systems courses, *Operating Systems Review* 21 (januari 1987), blz. 20-29.
26 Tanenbaum, A.S., S.J. Mullender en R. van Renesse (1986), 'Using sparse capabilities in a distributed operating system', *Proceedings of the Sixth International Conference on Distributed Computer Systems*, IEEE.
27 Tanenbaum, A.S. en R. van Renesse (1988), 'A critique of the remote procedure call paradigm', *Proceedings Euteco '88*, blz. 775-783.
28 Tanenbaum, A.S. en R. van Renesse, 'Distributed operating systems', *Computing Surveys* 17 (december 1985), blz. 419-470.
29 Van Renesse, R., A.S. Tanenbaum en A. Wilschut (1989), 'The design of a high-performance file server', *Proceedings of the Ninth International Conference on Distributed Computer Systems*, IEEE, blz. 22-27.
30 Van Renesse, R., A.S. Tanenbaum, H. van Staveren en J. Hall (1987), 'Connecting RPC-based distributed systems using wide-area networks', *Proceedings of the Seventh International Conference on Distributed Computer Systems*, IEEE, blz. 28-34.
31 Van Renesse, R., H. van Staveren en A.S. Tanenbaum (1989), 'Performance of the Amoeba distributed operating system', *Software – Practice and Experience* 19 (maart 1989), blz. 223-234.
32 Van Renesse, R., H. van Staveren en A.S. Tanenbaum (1988), 'Performance of the World's Fastest Distributed Operating System', *Operating Systems Review* 22 (oktober 1988), blz. 25-34.
33 Van Rossum, G. (1990), 'AIL – A class-oriented stub generator for Amoeba', in: J. Nehmer (ed.), *Proceedings of the Workshop on Experience with Distributed Systems*, Springer Verlag, 1990 (in preparation).
34 Welch, B.B. and J.K. Ousterhout (1988), 'Pseudo devices: user-level extensions to the Sprite file system', *Proceedings of the Summer USENIX Conference*, June 1988, blz. 37-49.

Andrew S. Tanenbaum, Robbert van Renesse, Hans van Staveren en Gregory J. Sharp zijn werkzaam bij de Faculteit Wiskunde en Informatica aan de Vrije Universiteit te Amsterdam.
Sape J. Mullender, Jack Jansen en Guido van Rossum zijn werkzaam bij het Centrum voor Wiskunde en Informatica te Amsterdam.
Hun E-mail-adressen zijn respectievelijk: ast@cs.vu.nl, cogito@cs.vu.nl, sater@cs.vu.nl, gregor@cs.vu.nl. Internet: sape@cwi.nl, jack@cwi.nl, guido@cwi.nl.

Dit onderzoek werd gedeeltelijk ondersteund door de Nederlandse organisatie voor Wetenschappelijk Onderzoek (NWO) onder nummer 125-30-10. Het onderzoek bij het CWI werd gedeeltelijk ondersteund door een subsidie van Digital Equipment Corporation.

Persbericht

vrije Universiteit amsterdam

Gedistribueerd computersysteem voor het eerst op de markt

Het onder leiding van informaticus prof.dr. Andrew S. Tanenbaum van de **Vrije Universiteit** ontwikkelde besturingssysteem **AMOEBA** zal op de markt gebracht worden door **ACE** Associated Computer Experts bv. Amoeba is een software-pakket waarmee honderden computers aan elkaar gekoppeld kunnen worden, dit terwijl het systeem zich voor de gebruiker gedraagt als één grote (super)computer. Amoeba is een van de eerste besturingsystemen waarmee computers effectief kunnen worden gekoppeld in zo'n gedistribueerd computersysteem. De marktintroduktie van Amoeba vond onlangs plaats door het tekenen van het contract tussen ACE en de VU. ACE heeft hierdoor de wereldwijde exclusieve distributierechten verworven voor Amoeba.

Het in Amsterdam gevestigde ACE Associated Computer Experts bv heeft sinds 1975 een leidende positie op het gebied van de ontwikkeling en levering van besturingssystemen in de internationale computerindustrie. In 1976 werd het besturingssysteem UNIX door ACE al succesvol commercieel toegepast. "Wij verwachten dat gedistribueerde besturingssystemen, net als open systemen nu, in de toekomst een belangrijke plaats in de computermarkt zullen krijgen" zegt Martijn de Lange, managing director van ACE, "en daarom is de keuze voor Amoeba, net als voor UNIX destijds, een logische stap in de strategie van ACE."

De ontwikkeling van gedistribueerde computersystemen is een gevolg van technologische vooruitgang. In de jaren '70 hadden de meeste organisaties slechts één, grote en dure, computer waar vele gebruikers op werkten ('mainframe'). Door sterke prijsdalingen ontstond in de jaren '80 de situatie dat elke gebruiker zijn eigen personal computer of werkstation kreeg, machines die vaak in een netwerk verbonden werden, waarbij enige communicatie mogelijk was.

Momenteel biedt het economisch gezien enorme voordelen (relatief goedkope) processoren aan elkaar te koppelen; het computersysteem wordt daarmee voor bepaalde toepassingen vele malen **sneller en goedkoper** dan een mainframe. De Amoeba-software maakt het mogelijk effectief op zo'n systeem te werken. De gebruikers van Amoeba leggen hierbij contact ('loggen in') met het gehele systeem in plaats van met een specifieke computer. Met andere woorden: voor de gebruiker gedraagt het systeem zich als een enkelvoudig conventioneel ('mainframe') systeem. Tanenbaum: "Behalve de voordelen van het werken op zo'n volledig geïntegreerd systeem is vooral de gunstige prijs/prestatie-verhouding een enorm voordeel. Daarnaast is het een volledig door ons zelf ontwikkeld produkt zodat voor het gebruik van Amoeba geen aanvullende licentieverplichtingen gelden, zoals voor UNIX het geval is".

De prestaties van het Amoeba-systeem zijn al met succes getest in de Europese ruimtevaartindustrie. Het wordt hierbij met name gebruikt voor het oversturen van digitale televisiebeelden, waarvoor een extreem hoog prestatieniveau vereist is. Tanenbaum verwacht dat de belangstelling voor Amoeba voornamelijk zal komen van bedrijven die veel machines willen integreren, bijvoorbeeld voor procesbesturing. Vooral onder Japanse (computer)fabrikanten bestaat veel belangstelling voor Amoeba, zo bleek tijdens een recente rondreis van Tanenbaum en een expert van ACE door Japan.

Noot voor de Redactie: ZIE OMMEZIJDE

Bureau Pers en Voorlichting

De Boelelaan 1107
1081 HV Amsterdam
Telefoon 020 - 5482671
Telex 11329 dpvvu

10.100.501

NOOT VOOR DE REDACTIE

Meer informatie over het Amoeba-systeem zal worden gegeven tijdens een bijeenkomst die zal worden gehouden op **vrijdag 7 juni 1991 om 10.00 uur in zaal R-224, faculteit Wiskunde & Informatica,** Vrije Universiteit, De Boelelaan 1081, Amsterdam-Buitenveldert. Prof.dr. A.S. Tanenbaum (VU) en dhr. M. de Lange (ACE) verzorgen een korte inleiding en een demonstratie van het Amoeba-systeem, waarna er gelegenheid is vragen te stellen. Voor aanmelding en meer informatie kunt u contact opnemen met: Gisela Plat of Richard Kroese (ACE), **tel. 020-6646416 / telefax 020-6750389.**

28.05.1991 / pb 91.44 / GvM

ACE Associated Computer Experts bv
Van Eeghenstraat 100
1071 GL AMSTERDAM
The Netherlands
Tel : +31 20 6646416
Fax : +31 20 750389
Telex : 11702 (ace nl)

associated computer experts bv

PERSBIJEENKOMST AMOEBA

Georganiseerd door de Vrije Universiteit Amsterdam en
ACE Associated Computer Experts bv

Datum: 7 juni 1991
Tijd: 10.00 uur
Locatie: Vrije Universiteit Amsterdam
Faculteit Wiskunde & Informatica
Zaal R-224
De Boelelaan 1081
Amsterdam-Buitenveldert

AGENDA

10.00 uur	Ontvangst met koffie
10.10 uur	Introductie
10.15 uur	Prof. dr. Andrew S. Tanenbaum (Vrije Universiteit) "Het gedistribueerd besturingssysteem Amoeba"
10.25 uur	Martijn de Lange (managing director van ACE) "Amoeba: technische en commerciële strategie"
10.35 uur	Gelegenheid tot het stellen van vragen
10.55 uur	Koffie
11.10 uur	Demonstratie van het Amoeba systeem door Prof. dr. Andrew S. Tanenbaum
11.30 uur	Einde bijeenkomst

Meer gedetailleerde informatie over het produkt Amoeba en de toepassingen zal tijdens de bijeenkomst voor u beschikbaar zijn.

associated computer experts bv

ACE Associated Computer Experts bv

Vrije Universiteit

Persbijeenkomst Amoeba

Amsterdam
7 juni, 1991

Prof. dr. Andrew S. Tanenbaum (VU)

Martijn de Lange (Managing Director van ACE)

Marco Roodzant (Marketing Director van ACE)

vrije Universiteit

ACE
associated computer experts bv

PERSBIJEENKOMST AMOEBA

Georganiseerd door de Vrije Universiteit Amsterdam en
ACE Associated Computer Experts bv

Datum:	7 juni 1991
Tijd:	10.00 uur
Locatie:	Vrije Universiteit Amsterdam Faculteit Wiskunde & Informatica Zaal R-224 De Boelelaan 1081 Amsterdam-Buitenveldert

AGENDA

10.00 uur	Ontvangst met koffie
10.10 uur	Introductie
10.15 uur	Prof. dr. Andrew S. Tanenbaum (Vrije Universiteit) "Het gedistribueerd besturingssysteem Amoeba"
10.25 uur	Martijn de Lange (managing director van ACE) "Amoeba: technische en commerciële strategie"
10.35 uur	Gelegenheid tot het stellen van vragen
10.55 uur	Koffie
11.10 uur	Demonstratie van het Amoeba systeem door Prof. dr. Andrew S. Tanenbaum
11.30 uur	Einde bijeenkomst

Meer gedetailleerde informatie over het produkt Amoeba en de toepassingen zal tijdens de bijeenkomst voor u beschikbaar zijn.

vrije Universiteit

associated computer experts bv

AMOEBA

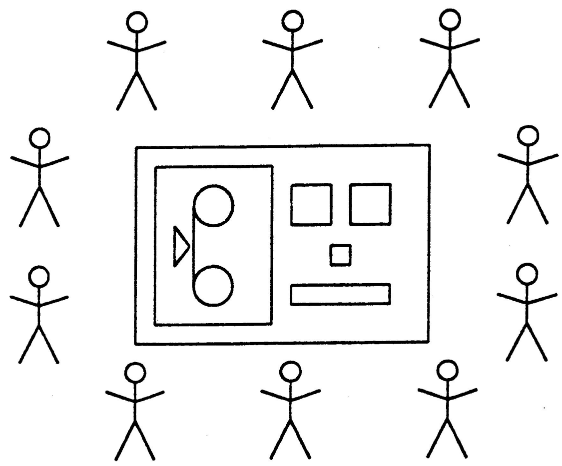
Prof. dr. A.S. Tanenbaum

Vrije Universiteit

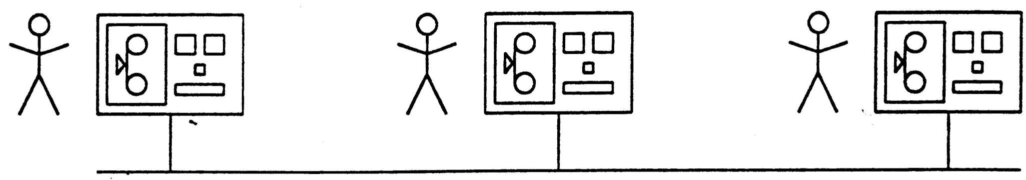
vrije Universiteit

SYSTEM MODELS

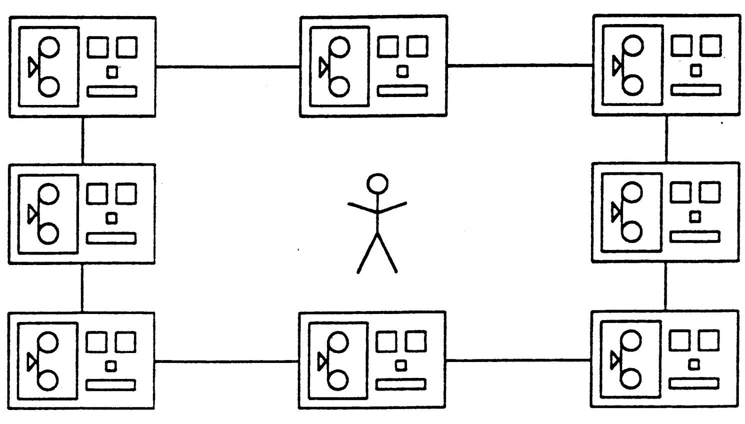
1970s



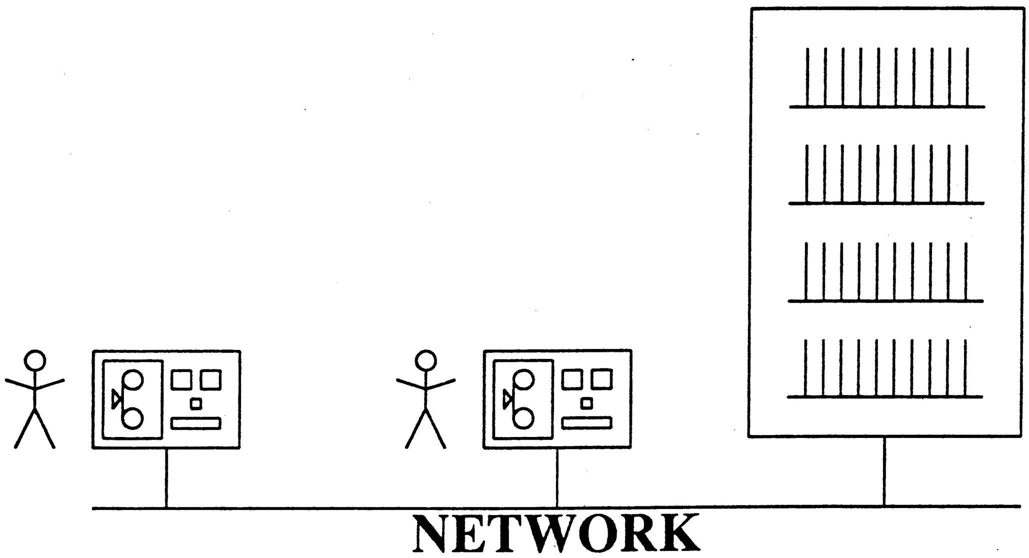
1980s



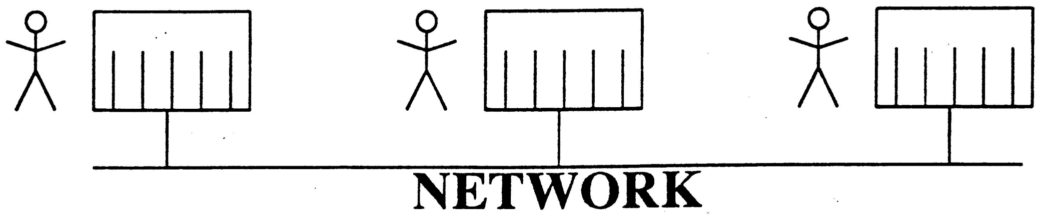
1990s



PROCESSOR POOL



PERSONAL MULTIPROCESSOR MODEL



PRICE/PERFORMANCE

System type	*MIPS*	*Price*
Mainframe	30	$10,000,000
32 Boards + I/O	64	$500,000

WHAT IS THE PROBLEM?

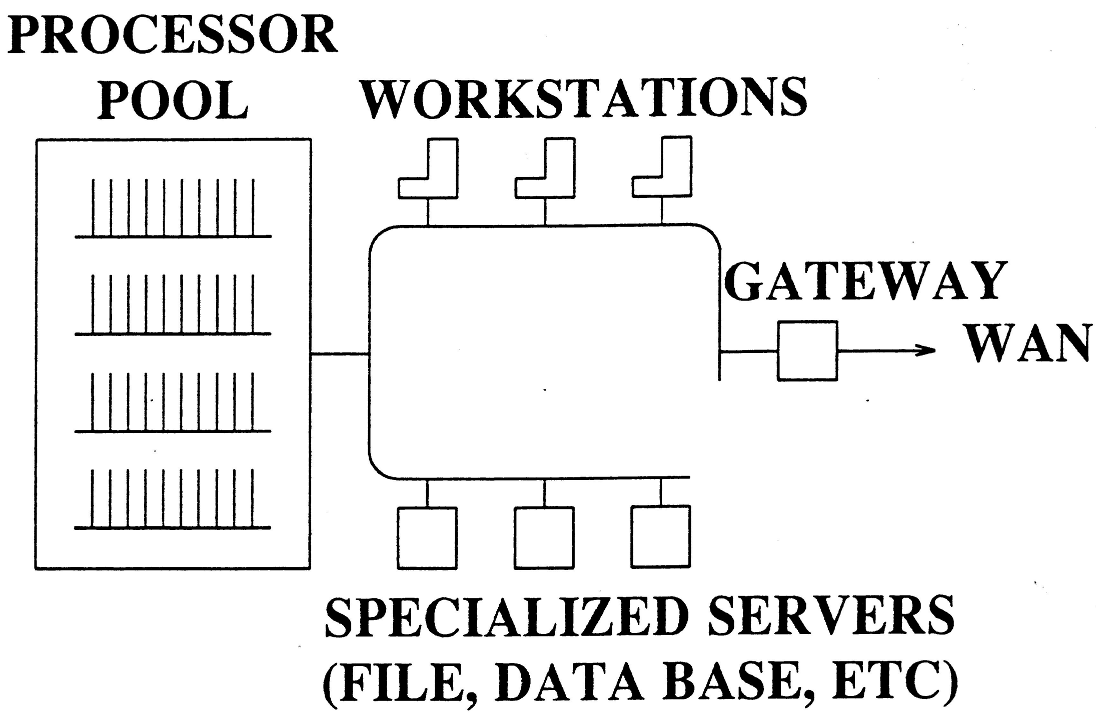
- No Software
  - No operating systems
  - No languages
  - No applications

OUR SOLUTION

- **Amoeba - An operating system for distributed and parallel computers**

ACE associated computer experts bv

AMOEBA SYSTEM ARCHITECTURE



PARTS OF AMOEBA

- MICROKERNEL

- SERVERS

  - FILE SERVER
  - DIRECTORY SERVER
  - TCP/IP SERVER
  - ETC.

AMOEBA SOFTWARE CONCEPTS

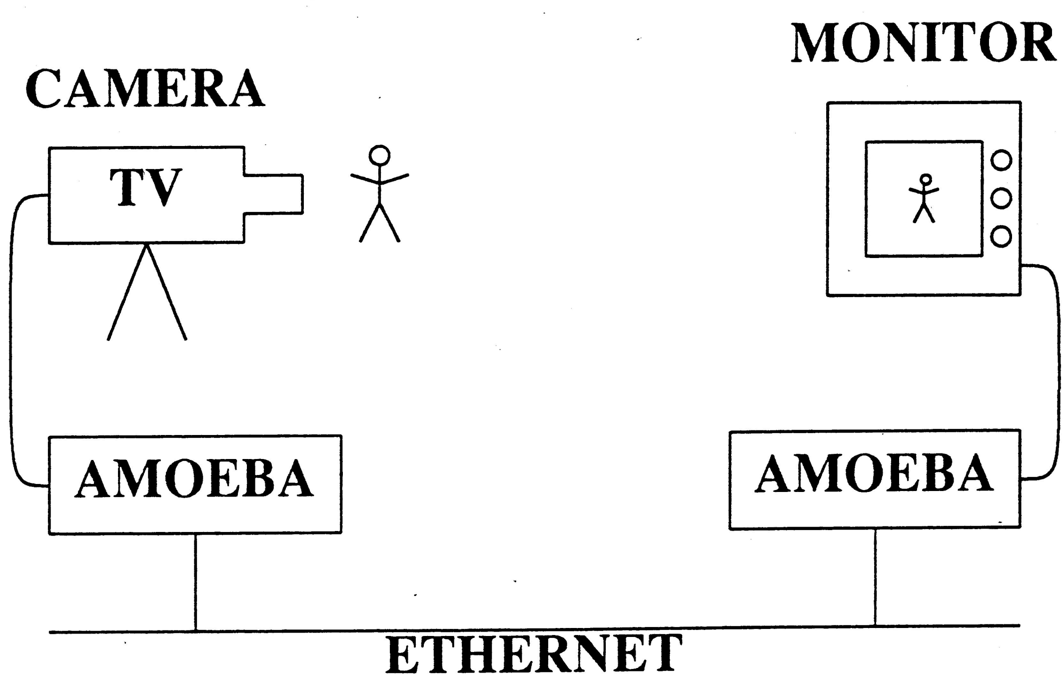
- **TRANSPARENCY**
- **OBJECTS**
- **CAPABILITIES**
- **REMOTE PROCEDURE CALL**
- **THREADS**

COMPARATIVE PERFORMANCE OF RPC

System	Hardware	Null RPC in msec.	Data Rate Kbytes/s	CPU MIPS
Amoeba	Sun 3/60	1.1	783	3.0
Cedar	Dorado	1.1	250	4.0
x-Kernel	Sun 3/75	1.7	860	2.0
V	Sun 3/75	2.5	546	2.0
Topaz	Firefly	2.7	587	5.0
Sprite	Sun 3/75	2.8	720	2.0
Mach	Sun 3/60	9.6	250	3.0

(Data from *Communications of the ACM*, Dec. 1990)

AN EXAMPLE: EUROPEAN SPACE INDUSTRY



APPLICATION - TRAVELING SALESMAN PROBLEM

- **FIND THE SHORTEST ROUTE VISITING EACH CITY EXACTLY ONCE**

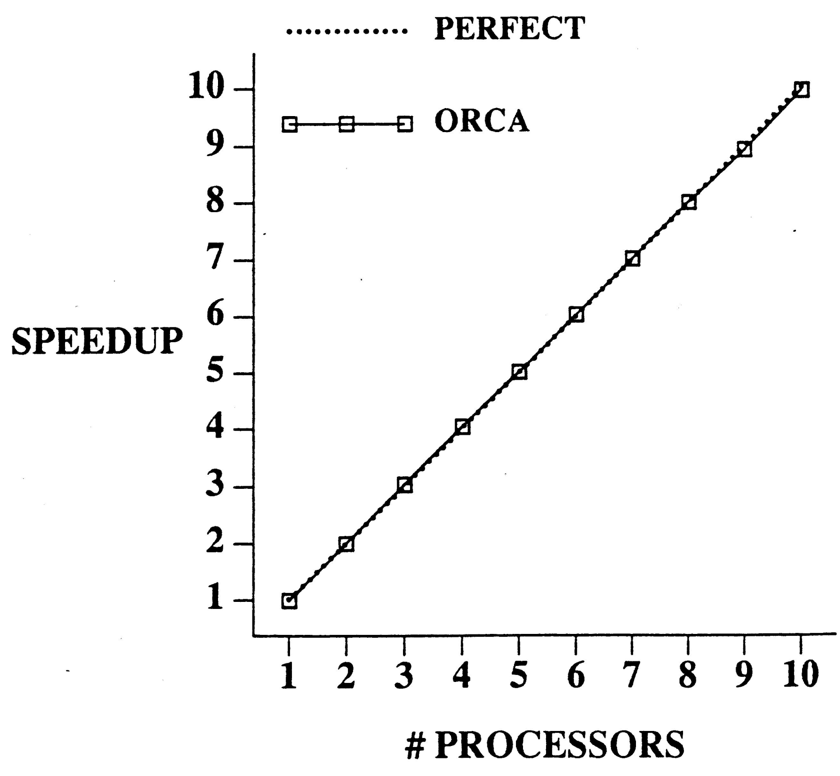
Amsterdam
New York
Tokyo
Rio de Janeiro
Nairobi
Stockholm
Sydney
San Francisco
Houston
Zurich
Singapore

CPU 1 works on paths starting: Amsterdam-New York
CPU 1 works on paths starting: Amsterdam-Tokyo
CPU 1 works on paths starting: Amsterdam-Rio de Janeiro
CPU 1 works on paths starting: Amsterdam-Nairobi
etc.

associated computer experts bv

PERFORMANCE OF TSP

- **Written in ORCA, Amoeba parallel programming language**



vrije Universiteit

FUTURE PLANS

- Amoeba 4.0 - Released Summer 1991

- Amoeba 5.0 - To be released Fall 1991
  - Support for broadcasting and multicasting
  - Support for new FLIP protocol
  - Automatic network reconfiguration
  - Support for SPARC architecture
  - Optimized ORCA compiler
  - Support for heterogeneous systems

- Amoeba 6.0 - Planned for 1992
  - Improved process managament with process migration
  - Virtual memory based on segments
  - File caching

ACE background 1975

- UNIX since 1976
- UNIX ports
- UNIX kernel improvements Europese + Japanse eisen
- Multi-processor UNIX
- UNIX Consultancy
- Consultants to X/OPEN
- Member of POSIX P1003
- Compilers (C, Pascal, F77, Modula-2, COBOL)
- UNIX networking
- Software Development Tools, CASE: CADESE
- UNIX validation and benchmarking 5%
- ESPRIT Projects:
  - COMPARE - Compilers for parallel chips
  - SPIRIT - High-end Technical Workstation

The Market Strategy

Amoeba is being targeted at several demanding market segments for distributed environments with high computing requirements, such as:

- Video image transfer and processing from satellite experiment set-up for ESA
- High speed physics data gathering and processing at CERN and NIKHEF

While it is being prepared to serve as the super-computing platform in environments with many (low-cost) processors in a pool:

- Many SPARCs, MC680x0's, MIPS, DEC, Intel386

All connected to Workstations for user interaction, such as:

- SUN-3's, SUN-4's, SPARC stations and DECstations

No competition with today's UNIX-alikes such as CHORUS and Mach, because these cannot cope with the speed requirements and transparent distribution over a massive number of processors the way Amoeba can.

The need for a Breakthrough

Rapidly growing demand for computing power and services cannot be met by a single (mainframe) machine because it is:

- too expensive
- difficult to maintain
- depreciates slower than chip-speed increases

It cannot be met by many separate or networked (PC) machines because they:

- create anarchy in the organization
- are a delight for floppy manufacturers
- are each too restrictive
- do not cope with application demands

Trends in OS Technology

ACE has carefully investigated the options for extension of its TAU/MU line:

- TAU (ACE) is X/Open standard UNIX
- MU (ACE) is highest performance (UNIX) for shared memory MP systems (e.g. SPIRIT)
- CHORUS and Mach (OSF) are either not fast enough or not really distributed
- UI (AT&T) is not multi-processor at all
- Amoeba is the fastest, transparently distributed and technically most sound

Where are we heading

The ever lower cost of hardware and ever higher cost of integration and software leads to:

- Scalable performance through scaling the number of processors at a much lower cost
- MIMD: massively parallel processing with 500 to 50.000 processors working in 1 'system'
- OS technology geared to hide the processors from the user
- MIMD machines replacing Super Computers: IBM and Cray are now changing their strategy
- Europe has a leading position in parallel processing   ESPRIT spinoff

The only OS really designed for this purpose is now going to the market: Amoeba

The Amoeba
Distributed Operating System

Andrew S. Tanenbaum

Peter Keuning

Release:	1.7
Date:	December 19, 1990
Status:	draft
Confidentiality:	private
Reference:	Amoeba

* UNIX is a registered trademark of AT&T in the USA and other countries.

The Amoeba Distributed Operating System

Andrew S. Tanenbaum

Vrije Universiteit
De Boelelaan 1081a, Amsterdam

Peter Keuning

ACE Associated Computer Experts bv.
van Eeghenstraat 100, Amsterdam

1. INTRODUCTION

At the beginning of the 1990s, it is becoming clear that the computing environment will continue to undergo fundamental changes, driven primarily by technological progress. Roughly speaking, we can divide the history of modern computing into eras:

1. 1970s: Timesharing (1 computer with many users)
2. 1980s: Personal computing (1 computer per user)
3. 1990s: Parallel computing (many computers per user)

In the 1970s and earlier, computers were huge, expensive, and located in computer centers. Most organizations had a single large machine.

In the 1980s, prices came down to the point where each user could have his own personal computer or workstation. These machines were often networked together, so that users could do remote logins on other people's computers or share files in various (often ad hoc) ways.

In the 1990s, we expect further price reductions to make it feasible to put many CPUs on a single board or even on a single chip. It will then become possible to build systems with tens or even hundreds of computers per user, possibly spread over a wide area. Such systems are usually called *parallel* or *distributed computer systems*.

This development raises the question of what kind of software will be needed for these new systems. To answer this question, a group under the direction of Prof. Andrew S. Tanenbaum at the *Vrije Universiteit* (VU) in Amsterdam (The Netherlands) has been doing research since 1980 in the area of distributed computer systems. This research, partly done in cooperation with the *Centrum voor Wiskunde en Informatica* (CWI), has resulted in the development of a new distributed operating system, called *Amoeba*, designed for an environment consisting of a large number of computers.

When Amoeba had reached the point where it was beyond the laboratory stage and was ready for commercial use, VU and CWI decided to join forces with ACE, a leading Dutch R&D company specializing in advanced software, including UNIX, CASE tools, compilers, and networking. While the research labs will continue to push the software technology forward, ACE will handle the marketing and most importantly, provide commercial support of Amoeba to customers requiring it.

For more detailed information please contact:

Contracting and Licensing:
Hester v. Marwijk kooy
Email: hester@ace.nl

Technical information:
Peter Keuning
Email: peterk@ace.nl

Commercial aspects:
Dick v. Brummen
Email: dvb@ace.nl

ACE - Associated Computer Experts BV
van Eeghenstraat 100
1071 GL Amsterdam
The Netherlands
Tel: +31 20 6646416
Fax: +31 20 750389
Telex: 11702 (ace nl)

2. DESIGN GOALS

The basic design goals of Amoeba are:

1. Distribution—Connecting together many machines
2. Parallelism—Allowing individual jobs to use multiple CPUs easily
3. Transparency—Having the collection of computers act like a single system
4. Performance—Achieving all of the above in an efficient manner

Amoeba is a *distributed* system, in which multiple machines can be connected together. These machines need not all be of the same kind. The machines can be spread around a building on a LAN, and multiple LANs can be connected together over wide-area networks (WANs).

Amoeba is also a *parallel* system. This means that a single job or program can use multiple processors to gain speed. For example, a branch and bound problem such as the Traveling Salesman Problem can use tens or even hundreds of CPUs, if available, all working together to solve the problem more quickly. Large "back end" multiprocessors, for example, can be harnessed this way as big "compute engines".

Another key goal is *transparency*. The user need not know the number or the location of the CPUs, nor the place where the files are stored. Similarly, issues like file replication should be handled largely automatically, without manual intervention by the users.

Put in different terms, a user does not log into a specific machine, but into the system as a whole. Once logged in, the user does not have to give special *remote login* commands to take advantage of multiple processors or do special *remote mount* operations to access distant files. To the user, the whole system looks like a single conventional timesharing system.

None of the above would be acceptable if the resulting performance were poor, so a major effort has been made to provide good performance. In particular, the basic communication mechanism has been optimized to allow messages to be sent and replies received with a minimum of delay, and to allow large blocks of data to be shipped from machine to machine at large bandwidth. These building blocks serve as the basis for implementing high performance subsystems and applications on Amoeba.

3. SYSTEM ARCHITECTURE

Since the hardware of the 1990s will no doubt be different from what was common in previous decades, it is worth while first describing the kind of hardware configuration for which Amoeba was designed. A typical Amoeba system will consist of four types of machines. First, each user has a workstation for running the user interface, X window System. This workstation can be a typical engineering workstation, or a specialized X terminal. It is entirely dedicated to running the user interface, and does not have to do other computing.

Second, there exists a pool of processors that are dynamically allocated to users as required. These processors can be part of a multiprocessor or multicomputer, be a collection of single-board computers (e.g., on a VME bus), or be a group of workstations allocated for this purpose. Usually, each pool processor has several megabytes of private memory, that is, pool processors need not have any shared memory (but it is not forbidden). Communication is performed by sending packets over the LAN. All the heavy computing happens in the processor pool.

Third, there are specialized servers, such as file servers and directory servers that run all the time. They may run on processor pool processors, or on dedicated hardware, as desired.

Finally, there are gateway machines that allow multiple Amoeba systems that are far apart to be connected together in such a way that to the user, the whole appears to be a single integrated system, rather than a collection of different systems.

All these components must be connected by a fast LAN. At present only Ethernet is supported, but ports to other LANs are negotiable.

4. THE AMOEBA MICROKERNEL

4.1 Microkernel + Server Architecture

Amoeba was designed with what is currently termed a *microkernel* architecture. This means that every machine in an Amoeba system runs a small, identical piece of software called the *kernel*. The kernel supports the basic process, communication, and object primitives. It also handles raw device I/O and memory management. Everything else is built on top of these fundamentals, usually by *server* processes running in user space.

Thus the system is structured as a collection of independent processes. Some of these are user processes, running application programs. Such processes are called *clients*. Others are *server* processes, such as the Bullet file server or the directory server. The basic function of the microkernel is to provide an environment in which clients and servers can run and communicate with one another.

This modular design makes it easier to understand, maintain, and modify the system. For example, since the file server is an isolated server, rather than being an integral part of the operating system, it is possible for users to implement new file servers for specialized purposes (e.g. NFS, database). In conventional systems, such as UNIX, adding additional user-defined file systems is totally infeasible.

4.2 Threads

In UNIX and many other traditional operating systems, a process consists of an address space and a single thread of control. In Amoeba, each process has its own address space, but it may contain multiple "threads of control" (*threads*). Each thread has its own program counter and its own stack, but shares code and global data with all the other threads in its process.

Having multiple threads inside each process is convenient for many purposes and fits into the model of distributed and parallel computing very well. For example, a file server may have multiple threads, each thread initially waiting for a request to come in. When a request comes in, it is accepted by some thread, which then begins processing it. If that thread subsequently blocks waiting for disk I/O, other threads can continue. Despite their independent control, however, all the threads can access a common block cache, using semaphores to provide interthread synchronization. This design makes programming servers and parallel applications much easier.

4.3 Remote Procedure Call

Threads often need to communicate with one another. Threads within a single address space can just communicate via the shared memory, but threads located in different processes need a different mechanism. The basic Amoeba communication mechanism is the *remote procedure call* (RPC). Communication consists of a client thread sending a message

to a server thread, then blocking until the server thread sends back a return message, at which time the client is unblocked. Amoeba has a special language called *Amoeba Interface Language* (AIL) for automatically generating stub routines to marshal parameters and hide the details of the communication from the users.

Because RPC is the basis of all communication between different machine in a distributed system, it has been optimized to be very fast. Measurements show that the time for a short RPC between two user-level processes on different machines (Sun 3/60s) connected by an Ethernet is 1.1 msec for the request plus the reply. Furthermore, the bandwidth for large transfers exceeds 800 kbytes/sec.

4.4 Memory Management

Not only are user processes structured as collections of threads communicating by RPC, but the kernel itself is as well. In particular, threads in the kernel provide access to memory management services. The Amoeba memory model is simple and efficient. A process' address space consists of one or more segments mapped onto user specified virtual addresses. When a process is executing, all its segments are in memory. There is no swapping or paging at present, although this may change in the future. The primary advantage of this scheme is simplicity and high performance. The primary disadvantage is that it is not possible to run programs larger than physical memory.

4.5 Input/Output

I/O is also handled by kernel threads. To read raw blocks from a disk, for example, a user process having the appropriate authorization, does RPCs with a disk I/O thread in the kernel. The caller is not aware that the server is actually a kernel thread, since the interface to kernel threads and user threads is identical. Generally speaking, only file servers and similar system-like processes communicate with kernel I/O threads.

5. SOFTWARE OUTSIDE THE KERNEL

The job of the Amoeba microkernel is to support threads, RPC, memory management and I/O. Everything else is built on top of these primitives.

5.1 Objects and Capabilities

Just as there are two unifying concepts in the kernel, threads and RPC, there are also two unifying concepts in the user-level software: objects and capabilities. An *object* is conceptually an abstract data type, that is, a data structure on which certain operations are defined. For example, a directory is an object to which certain operations can be applied, such as "enter file" and "look up name".

Amoeba primarily supports software objects, but hardware objects can also exist. Each object is managed by a server process to which RPCs can be sent. Each RPC specifies the object to be used, the operation to be performed, and any parameters to be passed.

When an object is created, the server doing the creation constructs a 128-bit value called a *capability* and returns it to the caller. Subsequent operations on the object require the user to send its capability to the server to both specify the object and prove the user has permission to manipulate the object. Capabilities are protected cryptographically to prevent tampering. All objects in the entire system are named and protected using this one simple, transparent scheme.

5.2 Bullet File Server

The standard Amoeba file server has been designed for high performance and is called the *bullet server*. It stores files contiguously on disk, and caches whole files contiguously in core. Except for very large files, when a user program needs a file, it will request that the bullet server send it the entire file in a single RPC. It is recommended to use a dedicated machine with at least 16 Mbyte of RAM for the bullet file server.

5.3 Directory Server

The bullet server does not provide naming services. It just reads and writes files, specified by capabilities. A directory server maps ASCII strings onto capabilities. Directories contain (ASCII string, capability) pairs; these capabilities will be for files, directories, and other objects. Since directories may contain capabilities for other directories, conventional hierarchical file systems can be easily built, but more general structures are also possible.

A directory entry does not need to contain only a single capability. It may contain a set of capabilities, to allow a file name to map onto a set of replicated files. When the user looks up a name in a directory, the entire set of capabilities is returned, to provided high availability. These replicas may be on different file servers, potentially far apart (the directory server has no idea about what kind of objects it has capabilities for, let alone where they

are located). Various operations are provided for managing replicated files in a consistent way.

5.4 Compilers

Amoeba comes standard with compilers for ANSI standard C, Pascal, Modula 2, and a parallel programming language called Orca. Each of these comes with appropriate libraries.

5.5 Utilities

Amoeba provides a large number of utilities modeled after the programs that come with UNIX. Furthermore, a number of new programs are provided such as *amake*, a highly parallel configuration manager.

5.6 UNIX Emulation

To aid in porting UNIX programs to the radically different Amoeba environment, the emulation library Ajax offers near-complete POSIX P1003.1 compatibility.

5.7 TCP/IP

Although the basic communication mechanism in Amoeba is the Amoeba RPC protocol, a user-level server is provided to allow TCP/IP communication, through RPCs to the TCP/IP server. In this way, remote machines can be accessed through Internet.

5.8 X Windows

Amoeba's user interface is the industry standard X Window System (X11R4). For X servers running on workstations, a special version of X is available that uses the Amoeba RPC for high-performance communication. When hard-wired X terminals are used, these can be interfaced using the TCP/IP server.

5.9 Connection to UNIX

A special UNIX driver is provided with Amoeba that can be linked into a UNIX kernel, allowing UNIX programs to communicate with Amoeba programs. It is also possible, as stated before, to use TCP/IP for this communication, but the feature described here is much faster and less complex. Utilities are provided to transfer files between UNIX and the bullet file server.

6. NONTECHNICAL ASPECTS OF AMOEBA

6.1 Source code availability

All Amoeba distributions contain the entire source code. There are currently no binary distributions available, but computer vendors and OEMs who license Amoeba may provide binary distributions to their customers.

6.2 Amoeba is unencumbered by AT&T licensing

Amoeba is a completely new operating system. Although it provides a partial UNIX emulation, it contains no AT&T code whatsoever. This means that no additional licensing is needed for Amoeba, and that the price for Amoeba is not increased because of the use of any kind of protected software.

6.3 Documentation

Amoeba comes with nearly 1000 pages of documentation, in several volumes:

- A collection of published scientific papers describing the initial basic ideas (the "yellow book")
- A users' guide (how to work with Amoeba; man pages for the utility programs)
- A programmers' guide (writing clients/servers; man pages for library routines)
- A system administrators' guide (how to install and maintain an Amoeba system)

6.4 Machines on which Amoeba runs

Amoeba currently runs on the following architectures:

1. Sun 3/50 and 3/60 workstations
2. 68020 VME-bus boards (Tadpole TP20 and TP22; Plessey; Motorola)
3. 68030 VME-bus boards (Force CPU-30)
4. VAXstation 2000, VAXstation II, MicroVAX 3500

In addition, ports are currently underway to the following machines. These ports will become available sometime in 1991.

1. Intel i386 (IBM AT bus)
2. SPARC workstation (Sun 4)

3. MIPS-based workstations (DEC 3100, SG IRIS)

Amoeba is normally distributed in "tar" format on either 1/4" QIC-24 streamer cartridge tape or on 1600 bpi 9-track magnetic tape. The distribution is about 20 Mbytes of source, documents and binaries. For each architecture a different subtree will be generated, and each binary tree will need another 80 Mbytes. The large size is due to the X libraries compiled into the binaries. The tape contains the Amoeba sources, documentation sources, native Amoeba binaries, and binaries of the utilities. The X sources are not included. However, the changes needed for the X11R4 sources are provided.

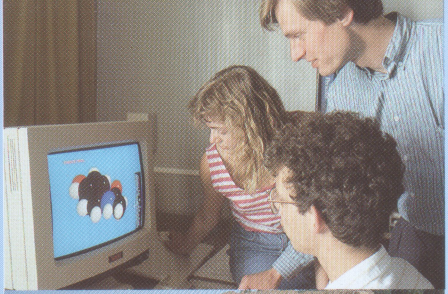
It should be noted that since Amoeba is a distributed system, it is necessary to have multiple machines to run it on. At the very least, there should be a dedicated file server machine, with at least 12M RAM, a processor pool with at least 1 pool processor (although more is highly desirable), and at least one workstation or X terminal for users. The pool processors can be single board computers or workstations, possibly without disks, monitors, and keyboards, since none of these are used. All machines must have an Ethernet connection.

For embedded applications, where the file server is not necessary and only the kernel is being used, it is possible to run the Amoeba kernel on a single CPU. Some installations are running Amoeba in kernel-only mode, in effect using it as a distributed high-performance kernel for real-time applications.

At the moment, Amoeba is initially compiled under UNIX; future releases will be able to boot directly from a cartridge.

vrije Universiteit

amsterdam







Onderzoek

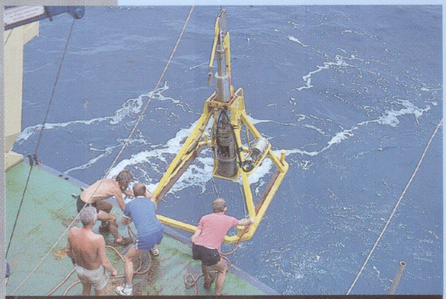
Het onderzoek dat plaats vindt aan de vijftien faculteiten van de Vrije Universiteit, omvat alle basisdisciplines die aan een universiteit thuishoren: van informatica tot psychologie en van geneeskunde tot economie. Onderzoekers van de Vrije Universiteit onderhouden contacten met vakgenoten over de hele wereld.
Belangwekkend onderzoek wordt verricht op tal van terreinen, onder meer op het gebied van de kankerbestrijding, moleculaire genetica, mariene geologie, ruimtelijke economie, ergonomie en oecologie. Naast de faculteiten telt de Vrije Universiteit enkele vooraanstaande onderzoekinstituten, waaronder het Instituut voor Milieuvraagstukken, het Economisch en Sociaal Instituut en de Stichting Onderzoek Wereldvoedselvoorziening. Nieuwe instituten die de aandacht trekken zijn het Instituut voor Extramuraal Geneeskundig Onderzoek en het Instituut voor Ethiek.

Ten behoeve van het onderwijs en onderzoek zijn op de universiteitscampus uitstekende faciliteiten aanwezig, zoals bibliotheken, laboratoria, werkplaatsen en eigen tv-studio's.
Een goed geoutilleerd congrescentrum met een capaciteit tot 1000 personen, maakt de Vrije Universiteit tot een erkende ontmoetingsplaats van wetenschap en samenleving.

De faculteiten van de Vrije Universiteit onderhouden tal van zakelijke contacten met het Nederlandse bedrijfsleven, de overheid en maatschappelijke organisaties en instellingen. De vraag naar contractonderzoek en andere vormen van zakelijke dienstverlening is de afgelopen jaren sterk gegroeid. De omzet in deze activiteiten bedroeg in 1989 ruim 50 miljoen gulden, dat is ongeveer 15% van het totale universiteitsbudget.

godgeleerdheid
rechtsgeleerdheid
geneeskunde
tandheelkunde
wiskunde en informatica
natuurkunde en sterrenkunde
scheikunde
aardwetenschappen
biologie
letteren
economische wetenschappen en econometrie
sociaal culturele wetenschappen
psychologie en pedagogische wetenschappen
wijsbegeerte
bewegingswetenschappen





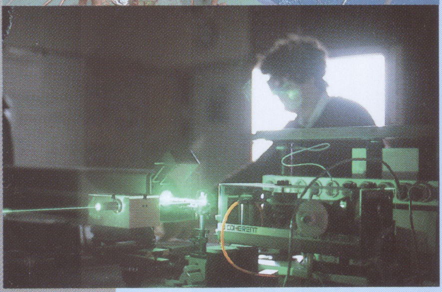


foto: S.O.Z.

Onderwijs

Kwaliteit en persoonlijke aanpak



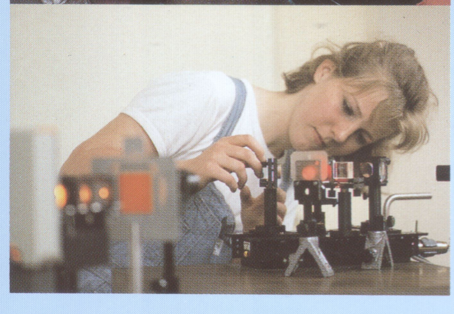
Met meer dan 40 studierichtingen biedt de Vrije Universiteit een ruime keus aan studiemogelijkheden. De opleidingen zijn van hoge kwaliteit, mede door een unieke manier van kwaliteitsbewaking en -bevordering. Het studieklimaat is goed en studenten hebben gemakkelijk entree bij de docenten. Er wordt gewerkt in naar verhouding kleine groepen en de studiebegeleiding heeft een persoonlijk karakter.
Veel studierichtingen doen mee aan internationale studenten-uitwisselingsprogramma's, die vooral in Europa plaatsvinden.
Binnen de gekozen studietrajecten krijgen de maatschappelijke mogelijkheden na de studie veel aandacht.

Inrichting van het onderwijs

Het universitaire dagonderwijs in Nederland is opgebouwd volgens de zogenaamde twee-fasenstructuur. In de eerste fase, die minimaal vier jaar duurt, krijgt de student een volwaardige universitaire opleiding die wordt afgesloten met een doctoraalexamen.
Voor enkele studierichtingen is er daarna een tweede fase, die hoog-uit twee jaar duurt en voorziet in een aanvullende beroepsopleiding, zoals voor arts, tandarts of leraar. Daarnaast heeft iedere faculteit een erkende onderzoekersopleiding (assistent-in-opleiding, A.I.O.).

Vele mogelijkheden





HBO-afgestudeerden die in hun vakgebied een universitaire titel willen halen, kunnen in vele studierichtingen een zogenaamd 'verkort doctoraalprogramma' volgen, dat circa twee à drie jaar duurt.
Naast het dagonderwijs zijn deeltijdopleidingen mogelijk. Wie overdag geen tijd heeft, kan 's avonds studeren. Voor ouderen of belangstellenden die geen volledige universitaire opleiding wensen, staat een scala van boeiende cursussen van hoog niveau open.
Afgestudeerde academici of andere hoger opgeleiden, die bij willen blijven in hun vakgebied, kunnen postacademisch onderwijs volgen.
Voor enkele beroepen zijn er postdoctorale beroepsopleidingen, zoals accountant, controller, EDP-auditor en (binnenkort) extern-organisatie-adviseur.
De Vrije Universiteit verzorgt ook contractonderwijs in opdracht van en betaald door bijvoorbeeld een bedrijf of een instelling.







foto: A.N.P.

Karakter

Gewoon Bijzonder

Als instituut voor de opleiding en vorming van academici en als wetenschappelijk bedrijf, is de Vrije Universiteit een universiteit als alle andere: een gewone universiteit.
De Vrije Universiteit is ook een bijzondere universiteit. Zij gaat niet van het Rijk uit, maar is ontstaan uit particulier initiatief.
Meer dan een eeuw geleden, in 1878, richtten orthodox-protestantse Nederlanders onder leiding van de theoloog en staatsman dr. Abraham Kuyper een vereniging op met als doel de stichting van een VRIJE christelijke universiteit. 'Vrij' betekende hier: vrij van de Kerk en vrij van de Staat, gebonden aan Gods Woord alleen. De oprichting van deze vereniging -en op 20 oktober 1880 van de Vrije Universiteit- was een antwoord op de achterstelling die men ondervond op tal van maatschappelijke gebieden; ook in het hoger onderwijs.

Veel studenten kiezen de Vrije Universiteit ook vanwege de uitstekende voorzieningen op het gebied van onder andere sport, cultuur en huisvesting.
Zo ligt op het nabijgelegen studentencentrum Uilenstede in Amstelveen het sportcentrum van de Vrije Universiteit. Voor wat betreft de recreatiesport is er een breed programma-aanbod.
Topsport wordt vooral beoefend in de sectoren basketbal en roeien. Op de Olympische Spelen van 1988 in Seoel, behaalden twee VU-studenten goud in de dubbel-twee. Maar ook veel topatleten in andere sporten studeren aan de Vrije Universiteit.
Het studenten-symfonie-orkest en het koor van de Vrije Universiteit behoren tot de beste amateur-muziekgezelschappen van Nederland.
Studenten kunnen meedoen aan gesprekskringen of leerhuizen, of aan het Vormingscentrum van de universiteit tal van praktische cursussen volgen.
Elke faculteit heeft wel een faculteitsvereniging die allerlei activiteiten organiseert, zoals lezingen, excursies en feesten. Daarnaast kent Amsterdam een groot aantal studenten-gezelligheidsverenigingen die elk jaar de deuren open zetten voor nieuwe leden.
In principe bestaan er ook mogelijkheden tot huisvesting op Uilenstede, waar in totaal zo'n 3500 wooneenheden worden beheerd.

Geloof en Wetenschap

Steeds open voor studenten van alle gezindten heeft de Vrije Universiteit zich ontwikkeld in de richting van een oecumenische universiteit die "God en Zijn wereld wil dienen, met als leidraad het Evangelie van Jezus Christus".
Vanuit deze intentie bestaat binnen de Vrije Universiteit een veelheid aan opvattingen over geloof en wetenschap. Leden van de universitaire gemeenschap (en velen daarbuiten) die actief participeren in de bezinning op de relatie tussen geloof en wetenschap, vinden in het 'Bezinningscentrum' van de Vrije Universiteit een stimulerende werk- en ontmoetingsplaats.
Met de eigen achtergrond van de universiteit hangt een organisatie-cultuur samen, waarin op eigen wijze met elkaar wordt omgegaan, samengewerkt en bestuurd. Daarbij zijn zorgvuldigheid, aandacht voor studenten en verantwoordelijkheid voor de medemens en de wereld uitgangspunt.
Ook worden bepaalde onderzoeksgebieden gestimuleerd, zoals ethiek, milieu, etnische studies en minderheidsvraagstukken en ouderenproblematiek. Het omvangrijke programma voor interuniversitaire ontwikkelingssamenwerking is gaandeweg een van de onderzoeks-zwaartepunten geworden. Zo wordt sinds jaren samengewerkt met universiteiten in Botswana, Lesotho, Swaziland, Mozambique, Indonesië en Zimbabwe. De Dienst Ontwikkelingssamenwerking van de Vrije Universiteit is het overkoepelend orgaan van deze activiteiten.

Vereniging

De vereniging, die nu 'Vereniging voor christelijk wetenschappelijk onderwijs' heet, benoemt bestuurders en hoogleraren en ondersteunt speciale projecten. Het VUSA Centrum van de vereniging (VUSA = Vrije Universiteit en Samenleving) organiseert jaarlijks tientallen cursussen in het land voor belangstellenden. De cursussen worden gegeven door medewerkers van de universiteit. Elke maand verschijnt het blad VU-magazine, met achtergrondartikelen over wetenschap, cultuur en samenleving vanuit een eigen kritische invalshoek. De universiteit ontplooit deze activiteiten omdat zij vanouds aan contact met de samenleving veel waarde hecht.

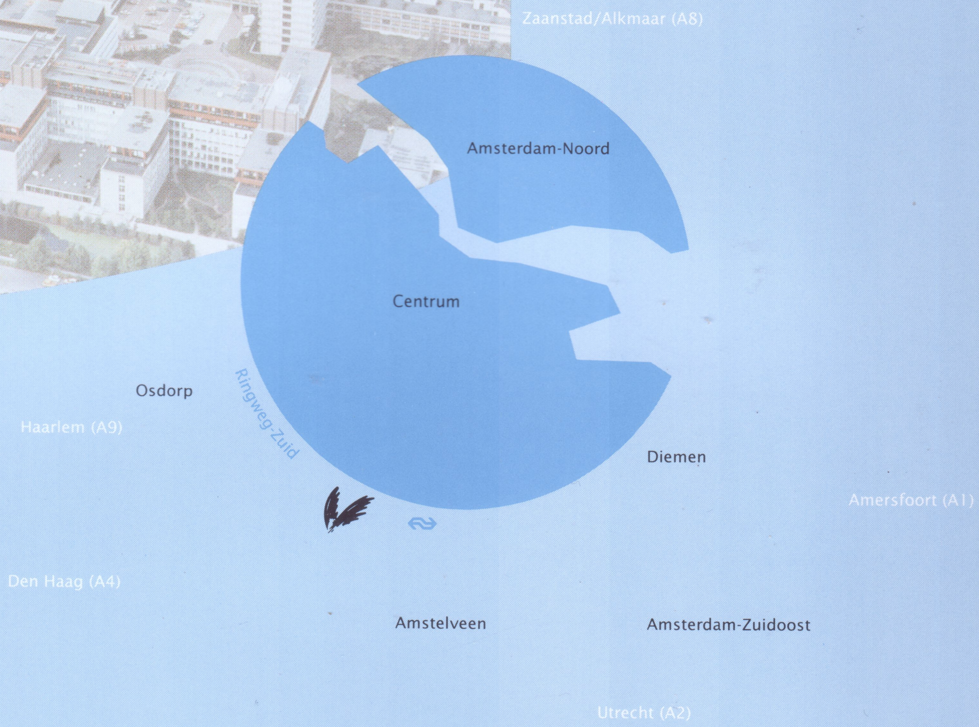




De Vrije Universiteit Amsterdam

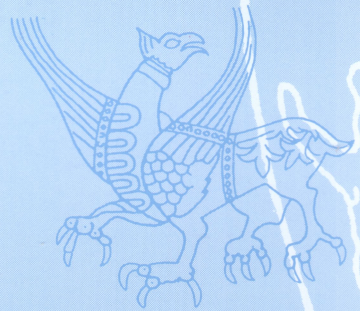
De Vrije Universiteit Amsterdam is een middelgrote universiteit met ongeveer 12.000 studenten en meer dan 2000 wetenschappelijke stafleden, waaronder ruim 300 hoogleraren. Daarnaast werken er circa 1700 personeelsleden in niet-wetenschappelijke functies.
De universiteit en het Academisch Ziekenhuis Vrije Universiteit zijn gehuisvest aan de De Boelelaan in Amsterdam-Buitenveldert, vlakbij de autosnelweg A 10 (Ringweg-Zuid). Een sneltram/metro, een stads-tram en diverse buslijnen zorgen voor uitstekende verbindingen met het centrum van Amsterdam en de wijde omgeving. Op loopafstand ligt het NS-station Amsterdam-Zuid/World Trade Centre.







Het beeldmerk van de universiteit, een griffioen, is een dier dat leefde in de fantasie van mensen in de oosterse en de klassieke oudheid.
Ook in de vroeg christelijke en middeleeuwse iconografie komt men het dier tegen.
Een griffioen heeft het onderlijf van een leeuw, kop en vleugels van een adelaar en de oren van een paard.
Het merk verwijst naar kenmerken van zowel het wetenschappelijk bezig zijn als van de Vrije Universiteit: staand in de werkelijkheid, nieuwsgierig, dynamisch en niet-voor-een-uitleg vatbaar.
*Het **vrije** van de universiteit is tot uitdrukking gebracht in de vormgeving van de vleugels.*











Bestuur en organisatie

Universiteitsraad en College van Bestuur

Het bestuur van de universiteit wordt gevormd door de Universiteitsraad en het College van Bestuur.
De Universiteitsraad bestaat uit gekozen afgevaardigden van de universitaire gemeenschap: studenten, wetenschappelijk personeel en technisch en administratief personeel, en uit leden die worden aangewezen door het bestuur van de Vereniging. De raad is belast met de algemene beleidsbepaling.
Het dagelijks bestuur van de universiteit is in handen van het College van Bestuur, dat daarin wordt bijgestaan door een ambtelijke dienst: het Bureau van de Universiteit.

Faculteiten en College van Dekanen



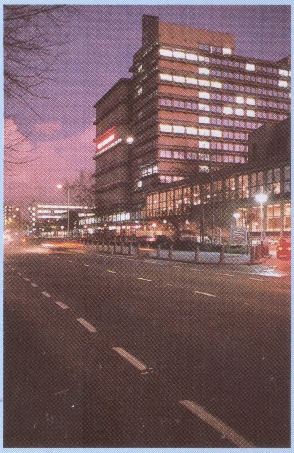
Alle studierichtingen aan de Vrije Universiteit zijn ondergebracht in de faculteiten. Elke faculteit bestaat uit een aantal vakgroepen die ieder een deelgebied van het onderwijs en onderzoek in de betreffende wetenschapsgebieden verzorgt.
De faculteiten hebben een grote bestuurlijke zelfstandigheid. Zij hebben een gekozen bestuur, dat wordt voorgezeten door de dekaan. De dekanen van de faculteiten vormen samen het College van Dekanen. Dit college geeft advies over onderwijs en wetenschapsbeoefening aan de hoogste bestuursorganen: de Universiteitsraad en het College van Bestuur.
De Rector Magnificus is voorzitter van het College van Dekanen en tevens lid van het College van Bestuur.

Nadere informatie

Algemeen
Informatiecentrum
Bureau Pers en Voorlichting,
hoofdgebouw, kamer 1D-03
telefoon: 020-548 3711

Opleidingen
Onderwijsvoorlichtingscentrum
Bureau Studentendecanen,
hoofdgebouw, kamer 0A-17
telefoon: 020-548 5000

Transferpunt
Wetenschapswinkel
Centrum voor Toegepaste Wetenschappen
Centrum voor Externe Dienstverlening
hoofdgebouw, kamer 1A-10
telefoon: 020-548 7051



Centraal Bezoek- en Postadres (hoofdgebouw)
Vrije Universiteit
De Boelelaan 1105
1081 HV Amsterdam

Telefoon
020-548 9222

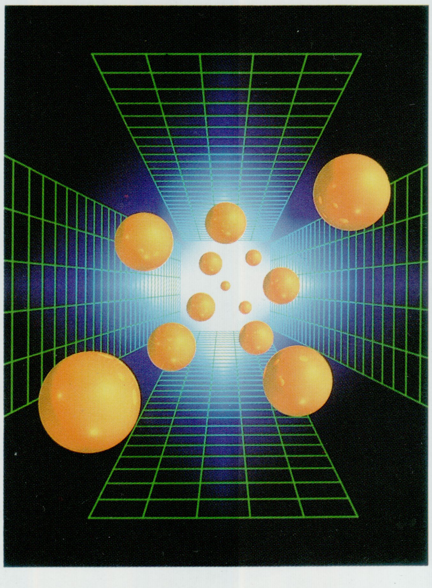
Bureau Pers en Voorlichting, 1991
Vormgeving en fotografie: Audiovisueel Centrum

THE SPIRIT PROJECT

The fastest Workstation: Made in Europe

Seven European partners build the new generation of supergraphics computers

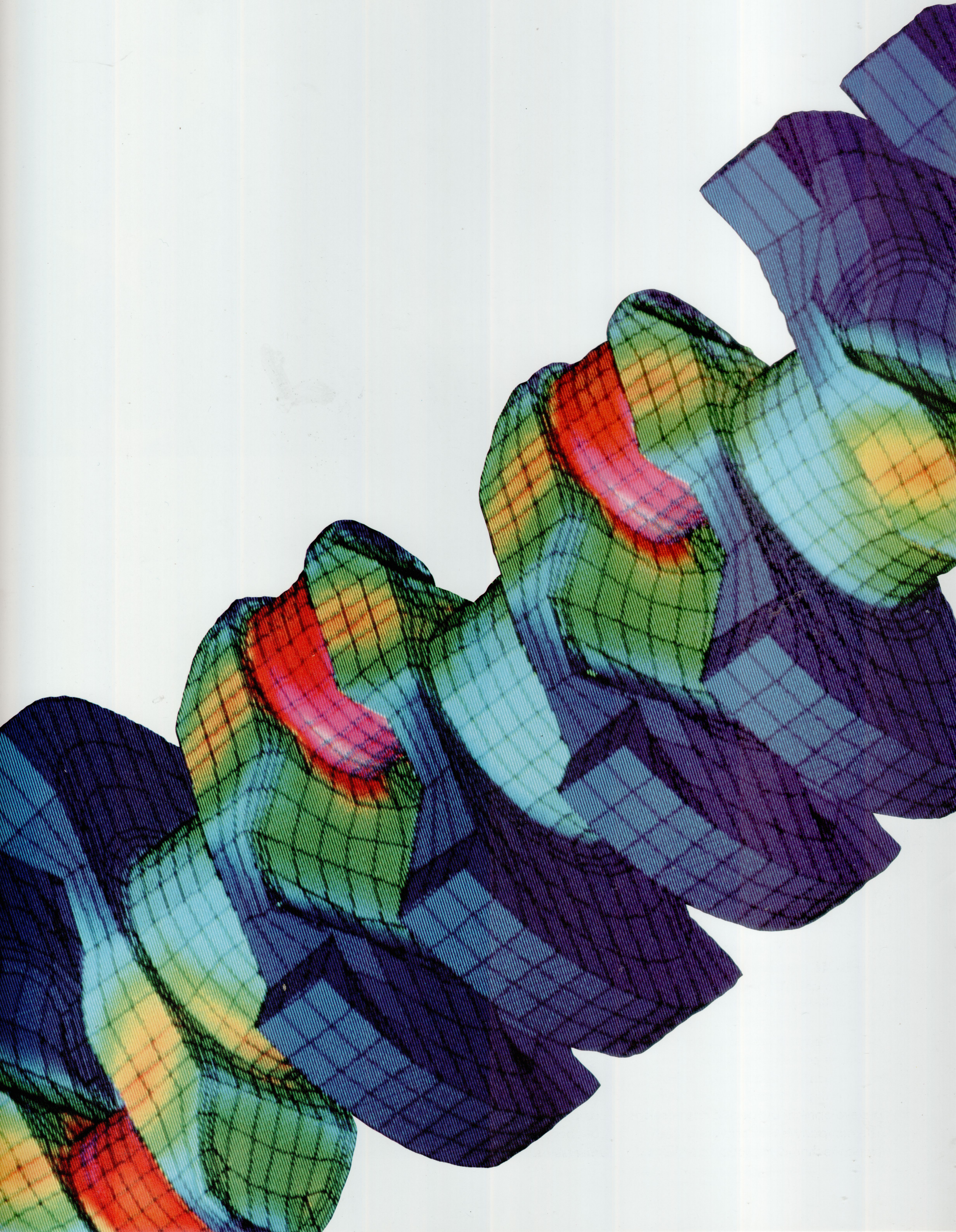
Almost two years of extensive research and development have been completed. Research prototypes have already been built. SPIRIT, designed to be the world's fastest workstation is nearing production. In close cooperation between universities and leading hi-tech companies a European supergraphics computer is being created, aimed at breaking the dominant market position of American and Japanese systems. SPIRIT goes beyond conventional concepts. SPIRIT is the first workstation to integrate multiple UNIX™ processors, interactive 3D graphics, and a novel Artificial Intelligence processor. The result is a workstation with extremely high performance – ideal for all graphics applications which need more than normal systems offer. SPIRIT will be employed in science, research and industry.

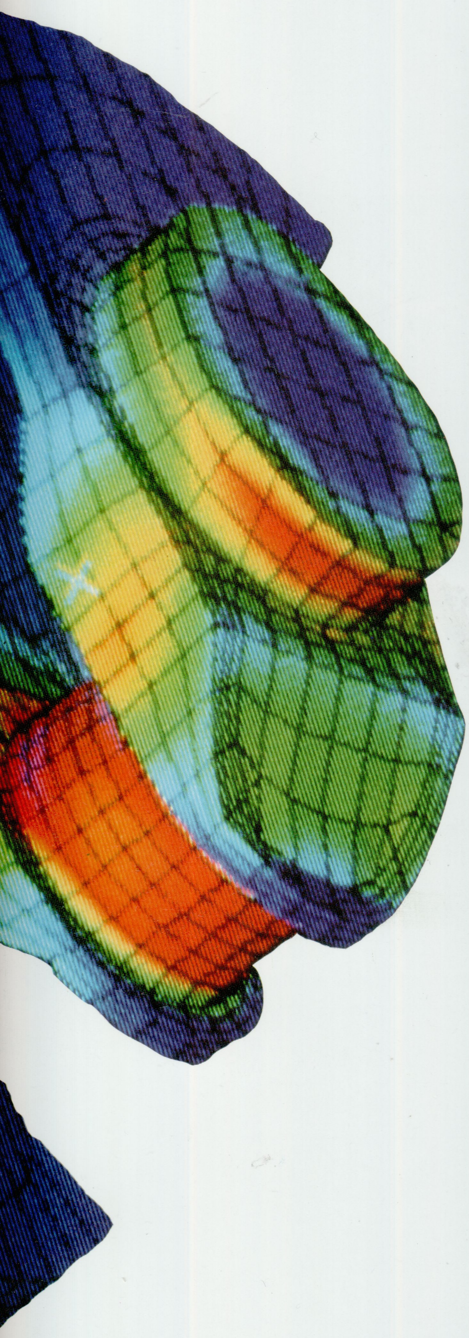


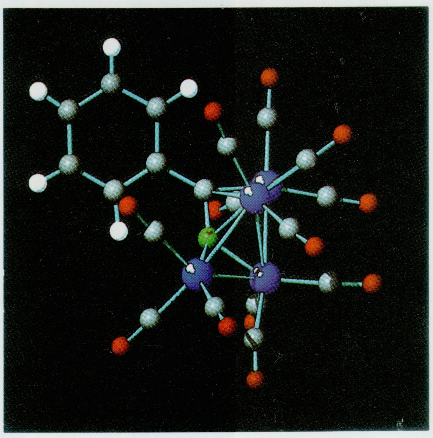
In a market with enormous development in the nineties Europe will have a product with the potential to gain a major share: SPIRIT.

SPIRIT makes Europe strong – the emergence of the workstation of the future resulting from the team-work of European universities and companies.

SPIRIT opens new dimensions in digital image processing. Faster. More precise. More intelligent.







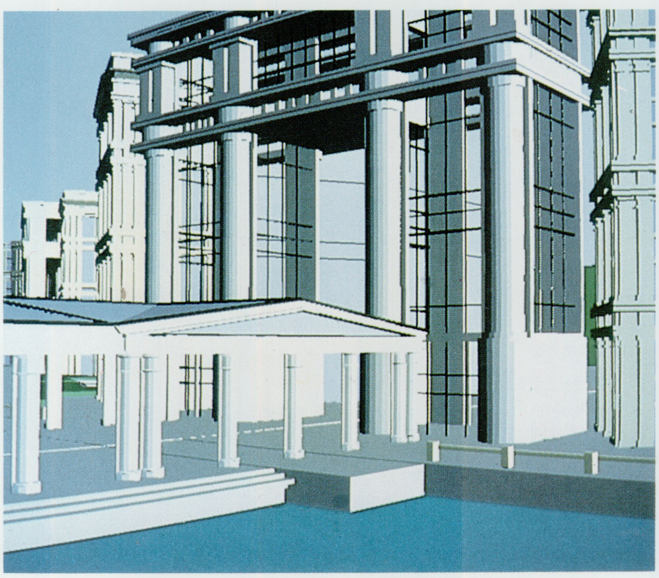
A class of its own

The supergraphics SPIRIT workstation sets a trend, which redefines the limits of computer graphics. SPIRIT provides photorealistic images, which can be generated and manipulated interactively. The quality of the images is achieved by full 24-bit colour and a high resolution display, which are utilised by state-of-the-art rendering algorithms.

A model for new applications

The market for supergraphics workstations will grow enormously in the nineties. First, new workstations will supersede old computers. Secondly, machines will be introduced in areas where graphical data processing is only rarely used at the moment, e.g. in fashion design.
In science and research workstations will simplify methods of analysis. Design and animation are further fields to be influenced by the capabilities of new systems such as SPIRIT.





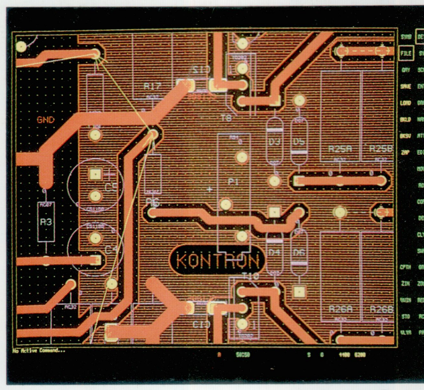
Ahead in the race for technology

At the moment the market for graphics workstations is dominated by American and Japanese companies. The realisation of the SPIRIT project allows Europe to build a significant share of the market as the modular open SPIRIT concept overtakes the power of competing systems. At the same time the price will be comparatively low. Today this potential is about to be realised. The launch into the global market will happen in the early nineties.

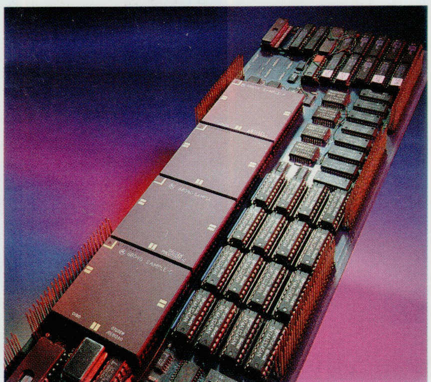
The Workstation of the future

SPIRIT is without precedence. The hardware and software has been conceived as a totally integrated design. Seven European R&D teams have contributed their knowledge and experience, with each partner bringing its special expertise to the project. This close cooperation has led to a system incorporating the latest technological achievements.

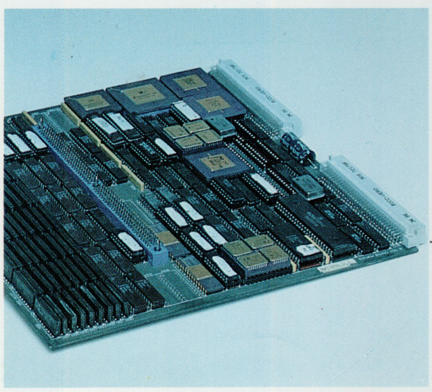
The operating system of the SPIRIT workstation is fully compatible with the industry-standard definitions for Open Systems based on UNIX™. The implementation allows heterogeneous operation of different types of processors at the same time in the same system. Special emphasis has been put on support of official and de-facto standards. Thus SPIRIT can be integrated seamlessly in technically oriented computing environments.

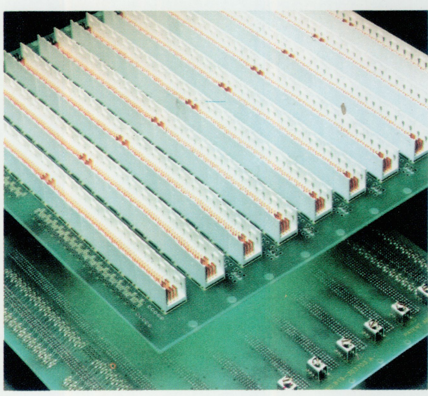


The processor boards of the SPIRIT workstation each house four 68040 CPUs running in parallel. Additionally the graphics subsystem and a dedicated AI processor for dealing with embedded artificial intelligence applications add processing power to the SPIRIT workstation. This multiprocessing capability gives a single workstation the power of a larger computer network.



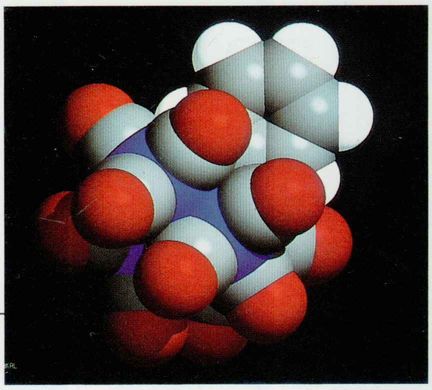
SPIRIT contains an innovative AI processor within the workstation. All the advantages of AI environments, e.g. expert systems, can be fully utilised. A dedicated processor is integrated to support the PROLOG language. It performs its operations at a speed in excess of one MegaLIPS (one million logical inferences per second), i.e. an order of magnitude more than conventional workstations.





The main bus of the workstation uses the high-performance Futurebus+ standard. Its standard interfaces allow a modular configuration of the CPU of 50 to 500+ million instructions per second. In addition a specially developed high-performance bus manages the synchronization and interprocessor communication.

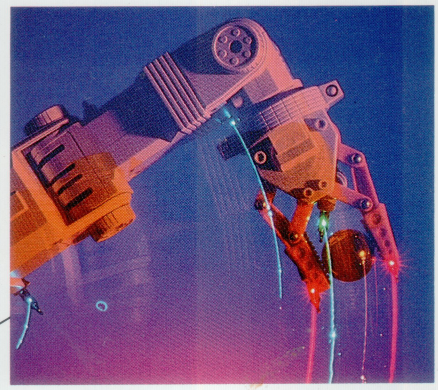




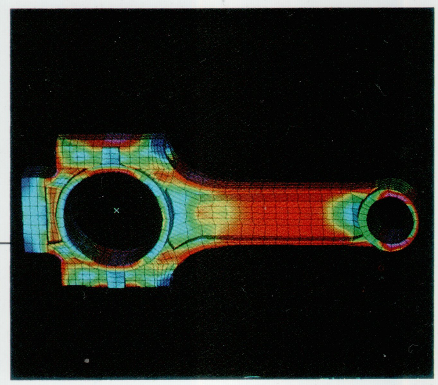
SPIRIT provides an advanced graphics environment. The graphics system is constructed on top of an extensible device independent Graphical Interface Layer, which is used to provide a platform for well-known industry standard environments. It provides a porting layer for the important 3D international standards, including PHIGS PLUS and the RenderMan interface. These systems support 3D modelling and rendering up to photo-realistic qualitiy. The parallel nature of the graphics subsystem provides the power for 3D animation.

For SPIRIT a special software environment has been designed. With it SPIRIT is universally programmable. The System supports all standard programming languages like FORTRAN-77, Pascal, C, ANSI-C, Cobol, MODULA-2, and Smalltalk-80. Smalltak-80 supports object oriented working and graphics prototyping. Individual applications thus can be written or adapted by any user without any problems.









In addition to the basic software turnkey solutions will be provided. For medical applications , for industrial requirements and for computer animation. The spectrum ranges from CAE to complex simulations.

The graphics subsystem includes a modern object processor and ASICs dedicated to scan-conversion. Because of the parallel and extensible architecture complex calculations can be performed in real time.

Seven qualified partners. An innovative

University of Sussex

- The design of the algorithms and architecture of the High-performance Graphics System of the SPIRIT Workstation.
- Detailed simulation of the Graphics System
- Contribution to the design of a special ASIC for fast rendering

The VLSI and Computer Graphics Group at the University of Sussex has an outstanding record of work in the field over many years. Their activities embrace distributed computing and graphics systems and the design of high performance chips. The group has an extensive record of industrial cooperation and has special expertise in the development of high-performance high-quality computer graphics systems.

KONTRON ELEKTRONIK

- Project Management
- Prototype production
- CPU board development
- Development of display controller and I/O subsystem

KONTRON ELEKTRONIK is owned by the car and technology group BMW and was founded in 1960. Among its wide-ranging products are image analysis systems, scientific

The University of Sussex

British Aerospace Dynamics

- Design of the AI processor

British Aerospace Dynamics is a part of the aviation and space technology group with the same name. Within this company the Dynamics division develops and produces electronic control systems for aviation and satellite communication. For these applications an AI system was engineered in 1988, which was twice as fast as similar systems, i.e. DLM 1. In the SPIRIT project Dynamics has worked on the next generation.

Eberhard-Karls—Universität, Tübingen

- Development of programmes for the generation, display and manipulation of 3 dimensional graphics
- Architecture of the image processor board and its components
- Design of highly integrated VLSI chips including the graphics ASIC

Under the directorship of Prof. Dr. W. Straßer the Institut für Informatik has worked in close cooperation with the industry for a number of years. In several research projects the scientists designed systems and components for graphics. Especially in the field of 3D graphics systems the institute gained international reputation.

Universität Tübingen

e concept.

N ELEKTRONIK

and technological electronics equipment and computers. The German company has developed and produced computers since 1974, image analysis systems and electronic equipment since 1979. The company has a world-wide workforce of 1100, 150 of them work in R&D. Because of its world-wide presence KONTRON ELEKTRONIK will look after the marketing and distribution of SPIRIT.

ACE
associated computer experts bv

ACE Associated Computer Experts bv

- Overall technical management
- Implementation of the heterogeneous tightly-coupled multiprocessor version of the UNIX™ operating system
- Integration of the EXPERT compilers
- Implementation of network protocols
- Systems architecture and QA/QC

The Dutch company ACE was founded in 1975 and is known as a specialist for operating systems and compilers. In 1976 they were the first European company to offer a UNIX™ development environment. As early as 1979 this was followed by their own distributed UNIX™ networking solutions, and soon thereafter their shared memory multiprocessor UNIX™ kernel.
In close cooperation with major computer manufacturers and companies ACE helped defining the standards known today as the X/Open CAE, and subsequently ACE implemented the first version of UNIX™ to adhere to these standards: TAU. Other ACE products are the EXPERT compilers, the CADESE configuration management package, the Supertest C validation suite, and the Benchmark suite.

QMW
UNIVERSITY OF
LONDON

Queen Mary and Westfield College

- Specification of the Graphics Interface Layer (GIL), which allows graphics software to be device independent.
- Extension of 3D graphics models for real-time interactive use.
- Provision of an object-oriented programming environment (Smalltalk-80).
- Extension of object-oriented concepts to support persistent objects and remote working

London University's Queen Mary and Westfield College has one of the largest Computer Science departments in the UK, and includes internationally-known researchers in the fields of graphics, object-oriented programming, and distributed systems.
The department has specialised research laboratories for Theory and Automated Reasoning, Advanced Computing Environments, Human-Computer Interaction, Distributed Systems, and Computer Vision. The internationally-known Centre for Parallel Computing works in association with the department.

CAPTION

CAPTION

- Design and specification of the graphics subsystem including both hardware and software
- Implementation of the object and image processor boards
- Specification and Coding of 3D graphics software
- Development of specific applications

The company is a subsidiary of the French technology group TELMAT.

CAPTION exclusively develops and produces graphics computers and programmes for 3 dimensional applications. Their hardware systems with integrated INMOS T800 transputers, Z buffers and modern image processors are known as the most powerful of their kind in the world. In the field of computer animation CAPTION has collaborated with several French universities.

Further information is available from:

EC
Mr. Michel Coomans
DG XIII/A/5 Brey 9/106
rue de la Loi 200
B-1049 Brussels
Phone: +32 2 235 12 62
Telefax: +32 2 236 30 31

KONTRON ELEKTRONIK GMBH
Dr. Rudi Wieczorek, Director of R & D
Breslauer Strasse 2, D-8057 Eching
Phone: +49 89 3 19 01-490
Telefax: +49 89 3 19 01-311

ACE ASSOCIATED COMPUTER EXPERTS BV
Martijn de Lange,
Managing Director
Van Eeghenstraat 100
NL-1071 GL Amsterdam
Phone: +31 20 6 64 64 16
Telefax: +31 20 75 03 89

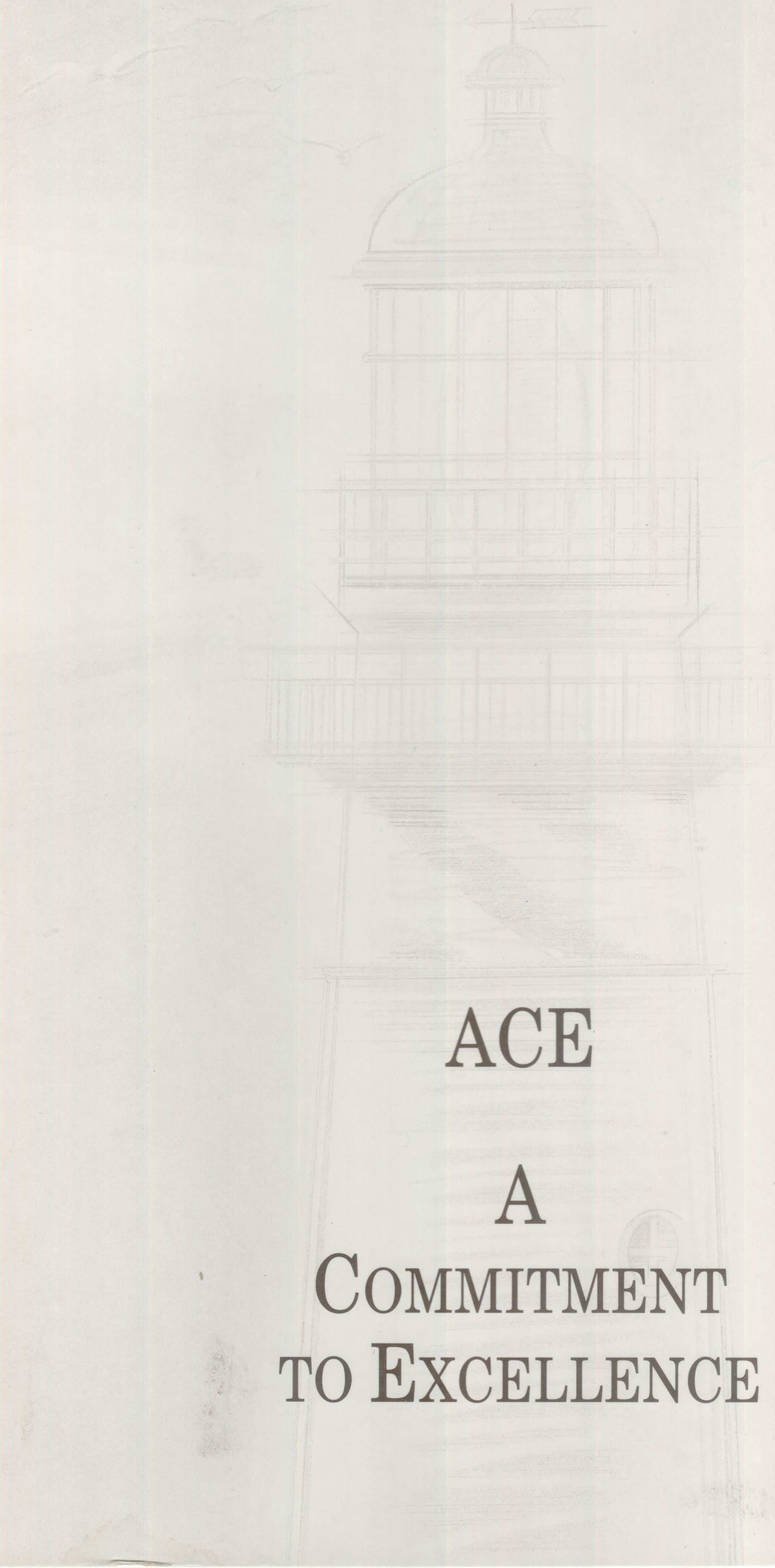
BRITISH AEROSPACE (DYNAMICS) LIMITED
Don Taylor
Clittaford Road
Southway
Plymouth
GB-Devon PL6 6DE
Phone:
+44 752 69 56 95-24 01
Telefax: +44 752 69 54 85

CAPTION
Patric Leclerc,
Managing Director
23, rue du Bignon
F-35135 Chantepie
Phone: +33 99 26 02 02
Telefax: +33 99 51 18 19

EBERHARD-KARLS-UNIVERSITÄT, TÜBINGEN WILHELM-SCHICKARD-INSTITUT FÜR INFORMATIK
Prof. Dr.-Ing. W. Straßer,
Director of Institute
Auf der Morgenstelle
10, C9
D-7400 Tübingen 1
Phone:
+49 70 71 29 63 56
Telefax:
+49 70 71 29 54 00

QUEEN MARY AND WESTFIELD COLLEGE
London University
G. Leonhard
Mile End Road
GB-London E1 4NS
Phone: +44 71 9 75 55 55
Telefax: +44 71 9 75 55 00

SCHOOL OF ENGINEERING UNIVERSITY OF SUSSEX
Prof. R. L. Grimsdale
GB-Brighton BN1 9QT
Phone: +44 273 67 80 47
Telefax: +44 273 67 83 99



ACE

A Commitment to Excellence

AN INTRODUCTION

ACE are specialists in the construction of software development environments. We develop technology which allows others to develop software, and we do it exceptionally well.

Our range of software is remarkable for its depth and breadth. Taken individually, each ACE product does what it was designed to do, excellently. Taken together, the ACE products combine to form a development environment that's remarkable for its productivity, maintainability, and elegance.

The cornerstone of ACE's software is The ACE UNIX, TAU. It's a full System V implementation which, as well as being X/Open compliant, has been reengineered to be multi-processor, faster, and more robust.

Next come the ACE EXPERT compilers. Widely recognised as the best compilers available, they compile from a number of development environments to a range of target architectures.

ACE also supplies a number of tools which significantly improve productivity in software development environments. These range from the ACE debuggers, through to the CADESE software management system and the EmROS electronic mail routing system.

And because software development is a competitive activity, ACE also provides both validation and benchmarking. ACE's independent status has led to the ACE Benchmarking Suite becoming the de facto industry standard for the evaluation of UNIX systems (benchmark results for over 100 systems are now available in book form), and has proved itself equally indispensable to hardware manufacturers, hardware purchasers and software developers.

And last but not least, ACE offers its expertise and experience in the form of consultancy. With over 300 man years in the forefront of software development (witness our participation in all the leading language and standardisation committees), we offer help and advice in all UNIX and UNIX-related areas; from operating systems, hardware design and compilers through to realtime systems, validation, networking and project management.

For example, one of our larger projects is the development of the SPIRIT workstation. Part of the ESPRIT programme, the SPIRIT project aims at building a 1000 Mips European workstation for the mid-90s, and represents a collaboration between seven of Europe's leading companies and universities. ACE are not only supplying the multi-processor kernel and utilities, we're also the technical leaders for the whole project.

Since its foundation in 1975, ACE has notched up a remarkable string of achievements. Such as being the first European company to supply its own UNIX implementation as a commercial product, developing the first UNIX Fortran 77 compiler to receive official validation, and being appointed as consultants to the X/Open group.

And although we're proud of our achievements and the recognition they've brought us, we've never been motivated by the desire to be in the public eye. Because, despite being pacemakers in one of the world's newest industries, ACE is driven by something much more old fashioned: the desire to produce software that's outstandingly well designed and outstandingly well put together. The dedication to produce software that functions, what ever the environment it happens to be in, as perfectly as man or machine can make it.

We call it a commitment to excellence.

INSIDE ACE

ACE is an unusual company. To begin with, every year we reinvest 30% of our revenue in R&D. Compare that to IBM (9.9 %), Ford (3.6 %), and Fujitsu (10.3 %), and you get some idea of how much importance we place on innovation and quality.

Nor is our recruitment policy exactly run of the mill: most companies recruit their staff as they arrive on the job market, but we believe in starting a little earlier. As a result of our close links with universities and research institutions, postgraduate and graduate students often spend time working at ACE as part of their studies. It's a relationship which is mutually beneficial to the universities, the students, and ACE, but it also has an important knock-on effect: the students get to know us, we get to know them, and if we like them, we offer them a job.

Of course, this is an excellent way of cutting recruitment costs, but the real benefit is much more substantial. In an industry in which the quality of what you produce is in direct proportion to the quality of your employees, we get the brightest and the best before anyone else even knows they exist.

Once they start at ACE, we provide a work environment which encourages hard work and creativity, rather than restricting it. To begin with, ACE is in the remarkable situation of having more UNIX systems than employees. More importantly, we don't insist that ACE developers stick to a conventional working day.



Experience has taught us that creativity and quality can't be shoehorned into the eight hours between 9am and 5pm, and so every ACE employee has his own key to the building. He can come and go as he (or she) pleases, and if he's in the middle of developing something and wants to work 24 hours at a stretch, that's fine by us.

And whereas most companies do their planning on the basis that each employee will work 1600 (out of 2000) hours a year, we do ours on the basis of 1200 hours. Which begs the question, what are ACE employees doing with the remaining 400 hours? Chatting, reading magazines, and drinking coffee?

Well, yes. You can't expect your staff to keep at the forefront of a fast-moving industry if they've always got their noses pressed to the grindstone. So we give ACE staff 400 hours in which to discuss their work with their colleagues (the best way of ensuring quality), to study, to read the trade magazines, and in general, to take time to simply think.

One of the most unusual aspects of ACE is our management structure: we call it Functionally Distributed Management (FDM). Just like any other development company, we allocate responsibility by product line; we have someone who's head of compilers, someone who's in charge of TAU, and so on. And, just like most good software companies, we also allocate responsibility per product part, so there's someone who's in charge of the Modula-2 front end, someone who's in charge of the Modula-2 documentation, and so on.

So far none of this is particularly unusual. But what makes FDM very unusual indeed, are two things. Firstly, everyone at ACE, however junior, has at least one area for which he or she is exclusively responsible: it's simply the best way we've found of maximising people's involvement in their work. Secondly, every responsibility carries with it a right of veto.

But if FDM sounds like a guaranteed recipe for constant conflict, it isn't. Because, in addition to the allocation of responsibilities already described, there is also a person within ACE who has the overall responsibility for ACE's development strategy. He functions, rather like the libero in a football team, by turning up wherever he is needed; and when a conflict does arise, it is ACE's libero who, taking into account ACE's overall strategy, casts the deciding vote.

The proof of the pudding is that it works. Our staff are involved, committed, and (not least because everyone is full-voting-rights shareholder) motivated.

And if FDM means that we spend more time in discussion than we would under alternative management structures, that's a price we're more than willing to pay. Because, as the French say, 'du choc des opinions jaillit la verité'.

TAU THE ACE UNIX

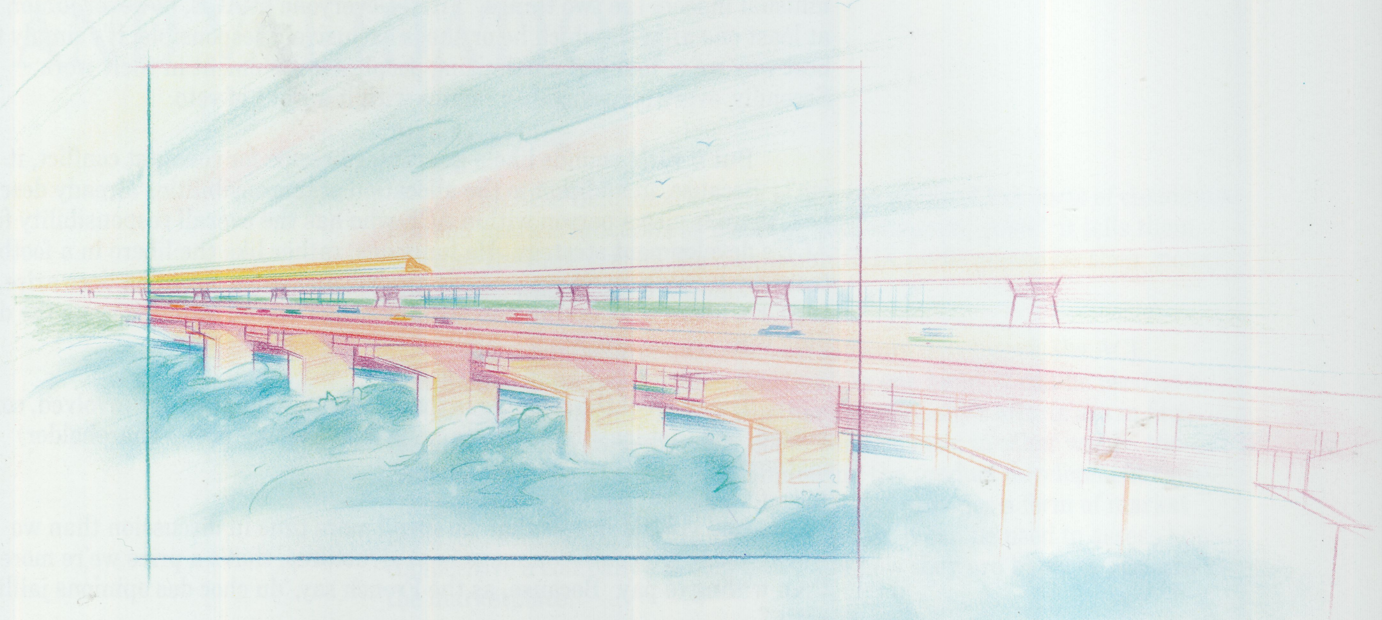
ACE has its own implementation of UNIX, and it's called TAU (The ACE UNIX). There are a number of UNIX implementations in the world, but we believe that TAU is clearly the best. Why? Well, there are two answers to this question, one short and one long. If you want the short answer, then you can skip to the last paragraph of this section; if you want the long answer, then we'll need some historical perspective.

UNIX was developed in the early seventies at Bell Labs in the USA. Over the next few years it grew rapidly until it became the leading operating system for research and development environments. But, and it's one of the many exceptional things about UNIX, it was not developed as part of a coherent and controlled plan; it grew in a piecemeal manner, added to and maintained by the people that used it.

And although UNIX in itself is an outstanding operating system, its existing implementations (System III, Version 7, Berkeley UNIX, System V etc), never really rid themselves of the legacies of UNIX's piecemeal development. In other words, they were internally messy, inefficiently coded, bug ridden, and time consuming to maintain.

And that's where ACE came in. We began in 1975 as the first commercial company in Europe to use UNIX; and we decided, given the state of existing UNIX implementations, that ACE could do better. If UNIX was to deliver its full potential, its weak parts had to be redesigned and rebuilt until the whole system was clean, fast, and efficient. Which is exactly what we set about doing with TAU.

To begin with, we made UNIX more reliable. We took out literally hundreds of bugs which were present in the original versions, and we redesigned the file system to make it more robust. We also improved the disk caching procedure and we added consistency checks so that the whole system continually monitors itself for correct operation.



We then went on to make our implementation faster by rewriting and optimising large parts of the kernel, by increasing memory allocation efficiency, and by reducing the computational overhead.

And to make sure that TAU would perform at its best, whatever the system's usage, we made both process switching and process management more tunable. And because reconfiguration is often a time consuming and error prone process, we simplified it by putting all the relevant information into one central, dynamic, data structure.

But we didn't stop there. We added a substantial number of new utilities and drivers. We added local and wide area networking based on TCP/IP, and we made sure that a multi-processor version was also available.

In other words, we took UNIX apart and we put it together again - but this time making sure that it was everything it should be. The result is an implementation of UNIX that is completely consistent with System V, but which is faster, more efficient, more portable, more reliable, and easier to work with. And not only that: in 1986 TAU became the first UNIX to fully conform to the X/Open portability guide.

All of which makes TAU the ideal UNIX implementation for software developers and computer manufacturers. Its X/Open compliance means not only that it can be used in multi-vendor environments, but also that it's easy to port applications to. Its logical and efficient construction means not only that hardware is better utilised, but also that the system administration overhead is lower.

Add to that an increased throughput and greater reliability, plus ACE's own set of tools and drivers, and you'll appreciate TAU's greatest benefit: your software engineers spend less time wrestling with the system, and more time doing what they're good at - developing software. No wonder the EEC has chosen the multi-processor version of TAU to be core software in the SPIRIT workstation.

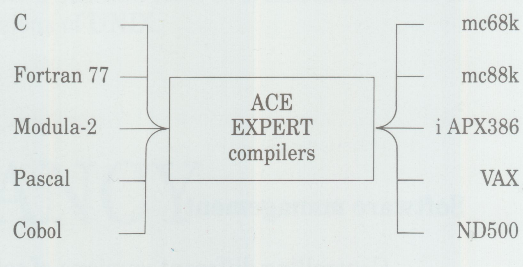
And that brings us to the short answer. The origins of UNIX itself may have been piecemeal, the origins of TAU are anything but. We believe that TAU is the best UNIX implementation available today, and for a simple reason. It was built that way, right from the start.

COMPILERS AND TOOLS

Compilers can be full of surprises. They can introduce new keywords and predefined identifiers, and incorporate extensions which a language's authors would never have contemplated. Or, in a bid to make it easier for themselves, they can take shortcuts, dropping a syntax check here and imposing a restriction there. Quirks like these often delay software development for a surprisingly long (and expensive) time.

As you would expect from a company with ACE's presence on most of the language standardisation committees, the ACE EXPERT compilers contain no such surprises. They are fast, reliable, and efficient. All our compilers are fully validated against the appropriate specification.

The ACE EXPERT compilers (cross) compile the most widely used programming languages, including C, Fortran 77, Pascal, Modula-2 and Cobol, from a number of development environments to a number of target architectures (see diagram).



ACE EXPERT compilers run on almost any UNIX system.



Built to a modular design, the EXPERT compilers are largely host and target independent, making them ideally suited to multi-vendor environments. Each language has a front end, a powerful intermediate code optimiser which is common to each language, and a target dependent code generator which produces the final code.

Programs written on a workstation can be ported without language conversion problems to the main system, just as easily as programs intended for a standalone system can be written and tested on a host system.

In addition, the common code generator allows users to intermix programming languages freely. For example, a Modula-2 program can call library routines written in C without affecting register usage or the calling sequences.

But the EXPERT compilers are not only flexible, they're also extremely powerful: sophisticated optimisers, applied over a five pass structure, ensure that the code produced by the EXPERT compilers is fast and effective, a fact borne out time and time again by benchmark measurements.

The initial level of the compilers is the language dependent front end. It parses the program source, checks syntax, and produces a language independent (identical for all languages) representation of the source. Before generating the machine dependent code, ACE's intermediate code optimiser goes to work, and it is at this point that the EXPERT compilers really come into their own. Compiler construction is perpetually evolving, and our intermediate code optimiser uses some of the most advanced techniques to impressive effect: even common optimisations, such as register allocation and copy propagation, increase in effectiveness when sophisticated analysis determines precisely where each optimisation will produce the greatest overall impact.

The assembly code produced by the code generator is then placed in a file and is (optionally) passed through an assembly optimiser which, using techniques like peep-hole optimisation, removes redundant instructions and reduces the image size. Where the code is being cross compiled, ACE also supplies a set of extra tools which produce a downline loadable image for the target system.

As you would expect from ACE software, the EXPERT compilers are highly reliable. In addition to being heavily tested by ACE both internally and at our beta test sites, they are checked against official validation suites and have more than 10,000 professional users worldwide.

ACE also provides sophisticated source level debuggers that are fully integrated within the EXPERT compiler family, and which support both native and cross development.

TOOLS

Software management

Controlling different versions of software is a difficult enough task at the best of times, but when products are cross compiled to a number of different environments, the administration problems become enormous. Enter CADESE.

Unlike most CASE tools, CADESE (Computer Aided Distribution Environment for Software Engineering) isn't solely concerned with the development of program sources. It provides a framework which supports the whole software cycle, from development, generation, and testing right through to identification and distribution. It even automates SPR (Software Performance Report) handling.

CADESE uses a version control system to maintain, in a highly compressed form, the source files for each official software release and ensures that earlier versions can always be regenerated. By closely integrating the version control system with the databases that support distribution generation and SPR handling, CADESE provides a unified structure and consistent format within which all aspects of software development are recorded, and preserves knowledge and expertise even in the face of personnel changes.

CADESE is a flexible and open-ended system, which makes it easier for development and support staff to use. Once installed, it will relieve your staff of the most laborious, time consuming and frustrating aspects of their jobs, and let them get on with their real work.

Electronic mail

Electronic mail systems are becoming increasingly important, particularly for those working in UNIX environments. However, the management of a mail system is a complex and time consuming task in which even minor errors can cause major problems. For those who don't see maintaining a mail system as their or their staff's main objective, the administration of a mail system can soon become a significant and undesirable overhead.

EmROS (E-mail ROuting System) provides simple management of complex electronic mail environments. Its hierarchical command interface (with command abbreviations and online help) provides an easy and powerful way of specifying routing and configuration information, and its improved (and cleaned-up) version of sendmail makes it extremely reliable.

It also provides extensive runtime support including logging, accounting, and access control, and is suitable for most standard transport systems. Mailers for non-standard transport systems can easily be added.

But EmROS not only makes things easier for the system administrator, it also makes it easier for mail users. It provides full Internet-style domain addressing independently of the transport system: all a user needs to know is the destination address, not how to get there. Goodbye hard!to!remember!path!user, and hello user@ address.

EmROS complies with all the relevant standards, such as RFC-821, RFC-822 and RFC-976, and is available for both Berkeley and System V versions of UNIX.

CONSULTANCY

Consultancy is a much abused word. In the last ten years it seems that the world and his brother have set themselves up as consultants, and more often than not their line of business has been something in the information sciences.

ACE, on the other hand, are in a totally different league. We began as leaders, and leaders we have remained. We were the first company in Europe to use UNIX as a software development environment and we were the first in Europe to supply an own UNIX implementation as a commercial product. We developed the first officially validated Fortran 77 compiler for the MC680X0, we carried out the first UNIX ports to MC680X0 machines, and we developed the first UNIX to comply with the X/Open portability guide.



Our prominent position in the world of development software is acknowledged worldwide. As technical consultants to the X/Open group we worked on the definition of the X/Open specifications, and we are the world's leading specialists in open systems.

Our membership of most of the important standards groups (such as those from ISO, IEEE, VITA, etc), besides ensuring that ACE is aware of potential developments even in their most embryonic stages, gives us an important platform from which to influence language and system standardisation in the direction we feel is best for the industry.

But if we have both the experience and the expertise (over 300 man years of it), we also have something more than that; we have the right approach. Because if ACE has been successful as consultants (and we have), it's due to the fact that our input as enthusiastic contributors to a project has been just as important as our input as experts.

Our consultants appreciate that 50% of effective consultancy is taking the time to listen carefully. They realise that successful consultancy is not so much about being a specialist, as being part of a team; about forming a close working relationship with clients so that the common goal is clearly understood and clearly achieved.

Today, a decade and a half after we began, ACE offers consultancy for all UNIX and UNIX-related areas; from system software, hardware design and compilers through to realtime systems, validation, networking and project management. For example:

Independent advisers

ACE's vendor independence means that we are able to offer impartial advice on a broad range of issues. A typical situation is where a new system has to be acquired. The company or institution in question is often in a difficult position: they know that the choice is critical to the company's future, yet they often lack the experience to take a fully informed decision.

The advantages of having ACE as an advisor are obvious. We're independent, so we can recommend which manufacturers to talk to. We know the right questions to ask and, just as importantly, we know how to interpret the answers. In short, our advice can often mean avoiding expensive mistakes.

UNIX systems

In addition to plain UNIX ports, ACE can check UNIX systems for compliance to the X/Open specifications, and where necessary adapt them so that they become X/Open compliant.

We can, and regularly do, advise on all aspects of system performance. In all the cases in which we have been called in, ACE quickly identified and solved the problem.

Operating system design

Over the years, ACE has designed and developed a number of operating systems for hardware manufacturers. We have a great deal of experience in the design of multi-processor and portable operating systems.

Hardware design

You might wonder what a software company like ACE can offer in the area of hardware design. But the fact is that our experience of writing operating systems and intelligent I/O drivers has given us valuable insights into what makes a good (or bad) marriage between hardware and software.

We have a lot of fruitful ideas about hardware and can often make significant contributions even at the design stage. For example, we have been heavily involved in the hardware design of the SPIRIT workstation, which includes a multi-processor architecture, high-speed busses and coherent cache protocols.

Compilers

ACE can advise on or assist in the implementation, testing and verification of compilers and programming language related products. For example, when CERN (Conseil Européen pour la Recherche Nucléaire) required a number of compilers which would comply with their object code format, we entered into a number of discussions to ascertain exactly what was needed. We were then able to deliver a product which exactly fitted CERN's requirements.

Networks

ACE offers expertise in all areas of networking, including LANs, TCP/IP, uucp, Ethernet etc. Amongst our networking projects have been operating systems for object oriented networks as well as transparent, message switching networks.

The central relay host for NLnet (the Dutch UNIX user group's network) uses EmROS, ACE's electronic mail system.

Realtime systems

As you would expect from operating system designers, ACE has an extremely strong background in realtime systems. About half ACE's technical staff have backgrounds in purely realtime environments.

Project management

ACE has extensive experience of managing both large and small projects. The SPIRIT workstation is one of ACE's larger projects. Its objective is the construction, using modular hardware, of an extremely high performance workstation (the SPIRIT workstation) to handle computer aided engineering, image processing, interactive graphics, simulations and AI. Running at speeds up to 1000 Mips, the SPIRIT will have a processing power geared to the mid-90s, 100 times more powerful than today's workstations. The SPIRIT workstation project is a collaboration between seven of Europe's leading companies and universities and is part of the ESPRIT programme.

ACE has the technical leadership for the whole SPIRIT workstation project, as well as being the developer and supplier of the multi-processor kernel and its utilities.

BENCHMARKING AND VALIDATION

Benchmarking

The ACE Benchmark Suite meets the needs of two sorts of customers. For end users it functions as a sort of 'good shopping guide': it tells them whether they are getting the maximum UNIX performance for their money and, if not, where to get it from.

For manufacturers, OEMs, and software developers, the ACE Benchmark Suite has a double function. On one hand, it can be used as a yardstick with which to assess whether the company's efforts are in fact resulting in concrete improvements in performance; in other words, the Benchmark Suite is a means of monitoring the product development cycle. On the other hand, the Benchmark Suite also provides a form of market research: because its results are comparative, it can be used to assess how competitive a company's products are and how they are likely to perform in the marketplace.

Over the years, ACE has benchmarked over 350 systems, including PCs, workstations and multi-processor superminis and, thanks in part to ACE's vendor independence, the ACE Benchmark Suite has become accepted as the de facto industry standard for evaluating UNIX systems. So much so in fact, that the results of the ACE Benchmark Suite are now published yearly under the title 'Benchmarking UNIX Systems'.

Benchmarking can either be carried out by ACE for you, or your company can purchase its own copy of the ACE Benchmark Suite. Of course, if the results of the test are to appear in 'Benchmarking UNIX Systems', then we will have to run the suite ourselves to ensure the authenticity of the results.

Validation

ACE has two groups of validation suites. The UNIX suite tests for completeness and validates according to the X/Open specifications (as well as non-conflicting SVID standards), whereas the compiler group has validation suites for C, Fortran 77, Modula-2, Pascal and Cobol.

These validation suites are thorough: they not only check for positive compatibility (ie. does a function work as it should), they also check for negative compatibility (ie. does the function return the right error value when forced to fail).

The SuperTest C validation suite is now the most extensive ANSI C validation suite in the world. Currently (December '89), it includes over 7,500 conformance and deviance tests, and over 425,000 stress tests.

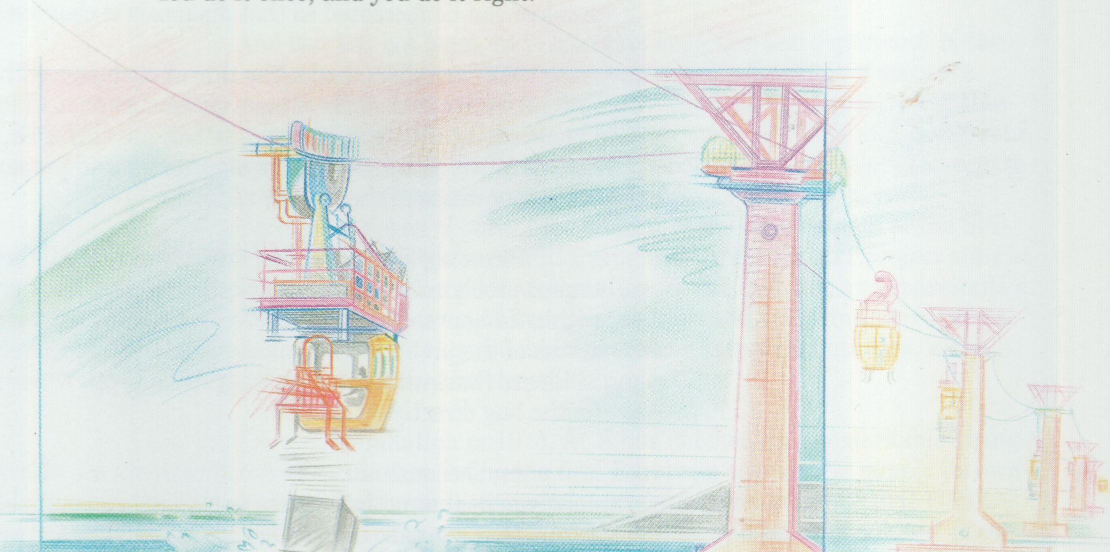
SUPPORT

Building it in

It's an old adage, but it's still true: the best support is the support that's already built in. At ACE, we build it in three different ways.

Firstly, the way we develop our software. ACE's whole approach is geared towards developing generic solutions rather than ad hoc ones. To a certain extent we build our software that way because it's practical - why build a situation specific solution when you can build an all purpose one? But to a certain extent we do it because it provides qualitatively better results.

When you're developing only one version of something, you can do it better. Not only because the discipline of writing portable, all purpose software means you write code that's more effective and more easily maintained, but also because you can afford to put that much more time and effort into it. You do it once, and you do it right.



Secondly, our testing is both extremely thorough and extremely comprehensive. Even before our products are distributed for three months beta testing, both at our beta sites and internally at ACE, they've been through an intense period of validation and evaluation. To give you an example: our compilers are not only expected to be able to recompile themselves, but also to be able to generate, flawlessly, a complete UNIX distribution.

And we are equally thorough when it comes to customising. An ACE customisation is never an ad hoc procedure; it's carried out according to a set of very strict rules which we have developed from our own experience. Each customisation is followed by a period of testing until the system is working perfectly; the details of the customisation, including the problems we encountered, are stored in a source tree so that knowledge, once acquired, is never lost.

Thirdly, we don't develop ACE products purely for the marketplace, we use them ourselves day in, day out. Of all the SPRs received over the last two years, more than 80% were generated internally by ACE staff (using the ACE SPR system) during beta testing. Which not only goes to show how much we use our own software, but also that no-one is a more demanding customer than we are. Resting on our laurels is something we don't do.

And of course, writing generic software not only means that we use it more intensively, but also that everyone else does as well. Take the ACE EXPERT compilers, for example. Their common code optimiser has now been used and tested in so many environments, cross and native, that it's as close as you can get to being 100% clean.

Support

When it comes to support itself, ACE applies the same principles we use in developing our software. We're structured, we're organised, and we've got our own generic solution for providing support. In other words, we use CADESE.

CADESE uses a database which holds every generation of every version ever supplied by ACE. The information is cross referenced per customer and allows ACE to completely reproduce any software release from an embedded version stamp and the release source tree. This implies that a customer's problem can be recreated so that diagnosis is clear and support effective.

ACE's support works like this. A customer with a problem uses the SPR generator and sends it to us via electronic mail (or mail, phone, fax etc). The incoming SPR is automatically fed into CADESE, which (using the information in the SPR) notifies the maintainer and prepares the data required for the regeneration of the correct version.

Incoming SPRs are then integrated into ACE's planning, with the most urgent problems receiving priority. Our report-back procedure ensures that, within 24 hours, the client receives an acknowledgment informing him of the status of the problem, the proposed fix and the delivery date of the fix. Urgent SPRs are fixed within 1-3 days of their arrival; where necessary, we can even fix the bug directly by modem.

And because our software developers are also our software maintainers, you deal directly with the person who'll be implementing the fix. Which means that you talk to someone who knows the software like the back of his (or her) hand, and you avoid the unpleasant situation where you have to wait on a third party before a fix can be implemented.

But, for ACE, fixing the bug itself is only half the job. Because, and this is part of our preference for structural solutions rather than quick 'n dirty patches, for every bug we fix we also write a test to check for it in future versions of the software. The test is added to the tests we perform for each new release, guaranteeing that bugs are effectively eliminated. When we fix bugs, they stay fixed.

A selection of organisations using ACE technology

AHOLD
Aeronautical Research Laboratories
Bull Sems
Commission of the European Communities
Commodore
Datex
Diab
Digital Equipment Corporation
Dolphin Server Technology
EGS
Ericsson
CERN
European Space Agency
GEC Avionics
Hewlett Packard
ICL
TNO
Instituut voor Kernfysica en Hoge-Energiefysica
Kontron
Motorola
Nixdorf Computer
Norsk Data
Olivetti
PTT telecommunicatie
Philips
Rijkswaterstaat
Schreiner Electronics
Siemens
Sony
Sun
Texas Instruments
Unisys

Throughout this document references are made mainly to 'he' and not 'she'. This is for convenience only and is not intended to have any particular significance.

For further information on ACE, please contact:

ACE Associated Computer Experts bv
Van Eeghenstraat 100
1071 GL AMSTERDAM
The Netherlands

tel: +31 20 6646416
fax: +31 20 750389
email: info@ ace.nl

Clemens van der Meer, Grafische vorm en realisatie


ACE
A
COMMITMENT
TO EXCELLENCE

EXPERT Compilers for Motorola Delta

EXPERT MC68000/010 code generator

EXPERT MC68020/030 code generator

EXPERT C front-end

EXPERT Fortran 77 front-end

EXPERT Modula-2 front-end

EXPERT Pascal front-end

The ACE EXPERT Compilers are an integrated family of reliable, high-performance compilers, available for the VMEexec development environment and for native use on the Motorola Delta Series 3000. The family consists of language-independent back-ends for MC68000/010 and MC68020/030 and a set of language-specific front-ends for various programming languages.

You need only purchase the relevant back-end, and can add front-ends for any required programming languages. Language support libraries are supplied with front-ends. Easy cross-language linking is a family-wide feature.

The back-end includes a highly optimizing code generator, extended COFF link-editor, assembler and all language-independent runtime libraries. Unless otherwise indicated, generated code supports the debugger XRAY68K under VMEexec and SDB in native use.

All components are now available for the Motorola VME Delta Series 3000 running SYSTEM V/68 release 3, version 5 or higher.

Compilers are delivered on 150 Mb streamer tape, and include complete documentation and a three month guarantee period. Maintenance contracts are available, providing for support by telephone and E-mail.

(data sheet release 1.14, dated 90/06/05)

Associated Computer Experts bv

Van Eeghenstraat 100
1071 GL Amsterdam
The Netherlands
Tel: +31 20 6646416
Tlx: 11702 (ace nl)
Fax: +31 20 750389

Currently, the following components are available:

Name EXPERT MC68000/010 code generator
Partnumber 056
Description Language independent part of EXPERT Compilers, generates code for MC68000 and MC68010 type of processors, running under VMEexec release 1.1.

Name EXPERT MC68020/030 code generator
Partnumber 057
Description Language independent part of EXPERT Compilers, generates code for MC68020 and MC68030 type of processors.
- Optionally generates inline MC68881/MC68882 code
- Assembler recognises complete MC68K line instruction set
- Generated code runs both under VMEexec release 1.1 and native on the Delta

Name EXPERT C front-end
Partnumber 058
Description Supports the C language as defined in Volume 4 of "The X/Open Portability Guide", Issue 2, January 1987.

Name EXPERT Fortran 77 front-end
Partnumber 059
Description Supports the ANSI standard X3.9-1987 and has been extended with the ISA bit manipulation features.

Name EXPERT Modula-2 front-end
Partnumber 060
Description Supports the language as defined in Wirths "Programming in Modula-2", editions 2 and 3.
- Comes with makefile generator that automatically determines module dependencies
- Modula-2 libraries supplied in source form
- XRAY68K support announced for 4Q90

Name EXPERT Pascal front-end
Partnumber 061
Description Supports ISO Standard Pascal level 1 (ISO 7185, ANSI/IEEE 770x3.97-1983) and optionally supports the Pascal MT+ dialect (dynamic string handling).

To order the compilers, contact your Motorola distributor and specify partnumbers of the components you need, whether these components should be covered by a maintenance contract, and CPU numbers of the machines on which you will use these components.

DISTRIBUTOR:

ACE Associated Computer Experts bv
Van Eeghenstraat 100
1071 GL Amsterdam
The Netherlands
Tel : +31 20 6646416
Fax : +31 20 6750389
Telex: 11702 (ace nl)

associated computer experts bv

THE ACE EXPERT™ COMPILERS - FACTS

The ACE EXPERT Compilers form an integrated family of reliable, high-performance compilers for C, Pascal, Fortran-77, Modula-2 and COBOL, designed by and for professional software developers. Uniform user interfaces to each compiler offer maximum flexibility through switch-selectable features (indicated as [S] below).

Each EXPERT Compiler consists of a front-end that is specific to the programming language, translating the program-source into a machine independent intermediate-code, a global optimizer operating on this intermediate-code, and a target specific back-end with target-optimizer, assembler and link-editor, for code selection, re-scheduling and target specific optimizations. Code compiled from different programming languages can be linked together.

The EXPERT Compilers emphasize conformance and efficiency. They also offer developers of UNIX® and Real-Time applications additional functionality such as: extra type checking, debugging, profiling, shared libraries and other versatile cross-development link-editor functionality.

Programming Languages (front-ends)

- C according to Volume 4 of "The X/Open Portability Guide", Issue 2, January 1987; ANSI-C available 1Q91
- Fortran-77 according to ANSI standard X3.9-1987 with the ISA bit manipulation extensions
- Modula-2 according to Wirth's "Programming in Modula-2", editions 2 and 3 [S]; comes with libraries in source form
- Pascal according to ISO Standard Pascal level 1 (ISO 7185, ANSI/IEEE 770x3.97-1983) and Pascal MT+™ [S] dynamic string handling
- COBOL according to ANSI standard X3.23-1974; support for RM™ level 2.0 C [S] and CIS™ [S]; ACCEPT and DISPLAY according to RM™ 85 [S]; ISAM runtime support
- Mixed language support

Global Optimizer (part of back-end) [S]

- Sophisticated execution frequency analysis based on Markov-theory
- Handles volatile and device registers [S]
- Handles asynchronous control flow
- Handles computed goto's
- Common subexpression elimination
- Constant and copy propagation
- Constant folding
- Strength reduction
- Register coloring and allocation

MC68K Target Architecture (back-ends)

- MC68000/010 code selection
- MC68020/030 code selection
- MC68040 code selection and emulation library
- MC68881/882 instructions in-line [S]
- Assembly optimizer [S]
- Assembler accepts all instructions of the MC68K family of processors and co-processors
- Efficient allocation of data, address and floating-point registers
- Branch optimization in assembler [S]
- Floating-point software support [S]
- COFF link-editor supports shared libraries [S] and segment relocation
- Debugging at source [S] and machine level
- Fully integrates with VMEexec™ environment with XRAY68K™ debugger
- Outstanding performance (see results overleaf)

Other Target Architectures (back-ends)

- Digital Vax
- Intel 386
- Norsk Data ND5000

Testing and QA/QC using

- ACE/SCO SuperTest™ C Validation Suite
- FSTC Fortran-77 Validation Suite
- BSI Pascal Validation Suite
- BSI Modula-2 Validation Suite
- FSTC COBOL-74 Validation Suite
- ACE Benchmark Suite
- E-mail based support and problem reporting

Documentation

- EXPERT Compiler documentation
- EXPERT Compiler manual pages
- On-line manual pages

OVERVIEW OF PERFORMANCE MEASUREMENTS

Selection from Benchmark Results					
		ACE EXPERT C Compiler	Standard C Compiler	Green Hills C Compiler	GNU C Compiler
Dhrystones	per second				
	Delta	**6521**	4538	6160	
	Sun	**5695**	4291		4058
	SONY	**10702**	8474		
	Olivetti	**4966**		4629	
Whetstones	per second				
	Delta	1517	526	**1533**	
	Sun	**1131**	895		971
	SONY	**1953**	1697		
	Olivetti	**895**		861	
Fibonacci	in seconds				
	Delta	**27.8**	40.9	35.3	
	Sun	35.2	**34.2**		48.2
	SONY	**20.5**	**20.5**		
	Olivetti	**43.9**		50.6	
X-Server	ratings				
X-Lines	Sun	**15570**	14351		
X-Arcs	Sun	**30464**	13615		
xStones	Sun	**14684**	13915		

System description:

Delta	System:	Motorola VME 3304 Delta System
	Hardware:	MC68030 + MC68882
	ACE C:	890.4, options -O -OI
	Standard C:	System V/68C, Release R3V5, options OPTIM=BOTH, FP=M68881, DBALIGN=YES
	Green Hills:	C-68000 1.8.4, options -O
Sun	System:	SUN 3/60-4-C
	Hardware:	MC68020 + MC68881
	ACE C:	890.4, options -O -OI
	GNU C:	1.35, options -O
	Standard C:	SUN OS 3.5, options -O -f68881
SONY	System:	SONY NEWS 1850
	Hardware:	MC68030 + MC68882
	ACE C:	890.4, options -O -OI
	Standard C:	NEWS-OS, Release 3.2
Olivetti	System:	LSX 3020B
	Hardware:	MC68020 + MC68881
	ACE C:	890.4, options -O -OI
	Standard C:	Green Hills compiler, C-68000 1.8.3A, options -O



AVC-VU
v.d. Boechorststraat 1
1081 BT Amsterdam
FOTOGRAFIE©
Datum: Nr.: 4916/20

Links: dhr. Martijn de Lange
(Managing director ACE)
Rechts: prof.dr. Andrew S. Tanenbaum
(Hoogleraar Informatica VU)